

FEE20

DOCUMENT OBJECT MODEL (DOM)

Markus Stolze
(Michael Gfeller)

DOM ÜBERSICHT

Inhaltsverzeichnis

- **DOM – Motivation**
- **DOM – JavaScript Grundlagen**
 - Quiz
 - Repetition
- **DOM**
 - Grundlagen
 - Suche / Selektion
 - Navigation
 - Whitespaces
 - Manipulation
 - Element Attribute (generell)
 - Events
 - Lifecycle
 - Element *Data Attribute
- **Templating**
- **Formulare**
- **Local-Storage**

Lernziele

Die Teilnehmer können...

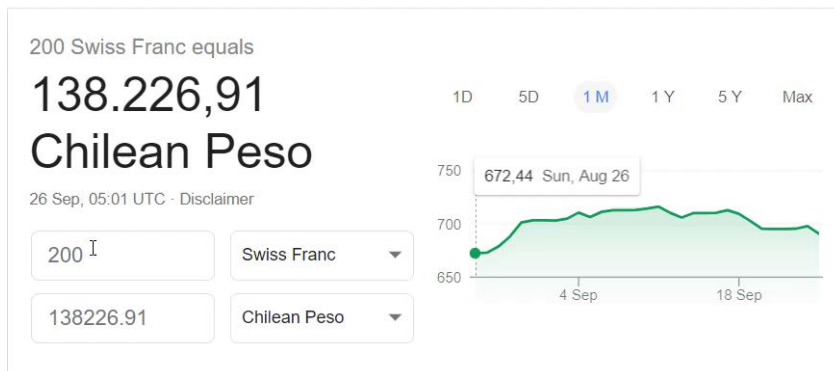
- erklären was die Design Idee des DOM ist
- auf unterschiedliche Methoden (unterschiedliche Funktionen/Argumente) Elemente im DOM selektieren
- auf unterschiedliche Arten das DOM verändern.
- Events an Beispielen erklären
- die Phasen des Event-Handling erklären und anwenden.
- HTML-Attribute in Kombination mit Event-Handling korrekt anwenden.
- DOM Lifecycle erklären und bei der Analyse von Fehlern anwenden
- Formularvalidierung mit JS optimieren
- Informationen zwischen zwei Seiten einer WebApp im Local-Storage teilen

MOTIVATION

Motivation

0

Rad		x!	()	%	AC
Inv	sin	ln	7	8	9	÷
π	cos	log	4	5	6	×
e	tan	√	1	2	3	−
Ans	EXP	xʹ	0	.	=	+



SBB CFF FFS

Von

Von I

Pflichtfeld, bitte ausfüllen.

Nach

Haltestelle, Ort, Sehenswürdigkeit

Datum

Zeit

< Mi, 26.09.2018 >

06:56

Ab ☐ An

Verbindung suchen →

Motivation

- **Interaktives HTML**
- **Interaktive Formulare**
- **Bessere Usability**
 - Fehlerbehandlung
 - Schnellere Antworten
- **Arbeit auf den Client auslagern**
- **Single Page Applications**

DOM JAVASCRIPT GRUNDLAGEN (REPETITION)

[01-dom-intro/00-js-quiz.js](#)

[01-dom-intro/01-js-rep4dom.js](#)

Quiz 1

```
const myObj = {myProp: 42};
```

```
myObj.myProp = 17;
```

```
const container = myObj;
```

```
container.myProp = 33
```

```
console.log("myObj.myProp: ", myObj.myProp);
```

```
// A: 42
```

```
// B: 17
```

```
// C: 33
```

```
// D: Error
```

```
// E: undefined
```

Quiz 2

```
const myObj2 = {myProp: 42};

myObj2.myProp = 17;

function changeMyProp(theObj) {
  theObj.myProp = 33;
}

changeMyProp(myObj2);

console.log("myObj2.myProp: ", myObj2.myProp);
```

```
// A: 42
// B: 17
// C: 33
// D: Error
// E: undefined
```

Quiz 3

```
const myObj3 = {myProp: 42};  
  
myObj3.myProp = 17;  
  
container3 = changeMyProp;  
  
container3(myObj3);  
  
console.log("myObj3.myProp: ", myObj3.myProp);
```

```
// A: 42  
// B: 17  
// C: 33  
// D: Error  
// E: undefined
```

Quiz 4

```
const myObj4 = {myProp: 42};

myObj4.myProp = 17;

function hof (theFunction, arg) {
  theFunction(arg)
}

hof(changeMyProp, myObj4);

console.log("myObj4.myProp: ", myObj4.myProp);

// A: 42
// B: 17
// C: 33
// D: Error
// E: undefined
```

Quiz 5

```
const myObj5 = {myProp: 42};
```

```
myObj5.myProp = 17;
```

```
myObj5.changeMyProp = () => this.myProp = 33;
```

```
myObj5.changeMyProp();
```

```
console.log("myObj5.myProp: ", myObj5.myProp);
```

```
// A: 42
```

```
// B: 17
```

```
// C: 33
```

```
// D: Error
```

```
// E: undefined
```

Quiz 6

```
const myObj6 = {myProp: 42};

myObj6.myProp = 17;

myObj6.changeMyProp = function () {
  this.myProp = 33;
}

myObj6.changeMyProp();

console.log("myObj6.myProp: ", myObj6.myProp);

// A: 42
// B: 17
// C: 33
// D: Error
// E: undefined
```

Quiz 7

```
function makeObj (val) {  
  let _prop = val;  
  const incProp = () => ++_prop;  
  return {incProp};  
}  
  
const myObj7 = makeObj(42);  
  
myObj7.incProp();  
  
myObj7._prop = 17;  
  
console.log("myObj7.myProp: ", myObj7.incProp());  
  
// A: 42  
// B: 44  
// C: 17  
// D: 18  
// E: Error
```

Quiz 8

```
function makeObj (val) {  
  let _prop = val;  
  const incProp = () => ++_prop;  
  return {incProp};  
}  
  
const myObj8 = makeObj(42);  
console.log("myObj8.myProp: ", myObj8.incProp(), "  
myObj7.myProp: ", myObj7.incProp());  
  
// A: myObj8.myProp: 43, myObj7.myProp: 45  
// B: myObj8.myProp: 45, myObj7.myProp: 46  
// C: Error
```


JS-Rep 1: Variables can hold Objects

```
// **  
// ** Variables can hold objects  
// **  
const demoObj = {name: 'theDemoObj', hidden: false};
```

JS-Rep 2: Object Properties can be Queried

```
// **  
// ** Object properties can be queried  
// **  
console.log('name of demoObj is: ', demoObj.name);  
    // output: "name of demoObj is: theDemoObj"  
  
console.log('demoObj: ', demoObj);  
    // output: "demoObj: { name: 'theDemoObj', hidden: false }"
```

JS-Rep 3: Object Properties can be Changed

```
// **  
// ** Object properties can be changed  
// **  
demoObj.hidden = true;  
console.log('demoObj: ', demoObj);  
// output: "demoObj: { name: 'theDemoObj', hidden: true }"
```

JS-Rep 4: Variables holding Non-primitive Values are "just" References

```
// **  
// ** Variables holding non-primitive values are "just" references  
// ** Different variables can reference the same object  
// **  
const anotherVarReferencingDemoObject = demoObj;  
console.log('anotherVarReferencingDemoObject: ',  
             anotherVarReferencingDemoObject);  
    /// output: "anotherVarReferencingDemoObject:  
    ///           { name: 'theDemoObj', hidden: true }"
```

JS-Rep 5: Functions are 'First Class Citizens'

```
// **  
// ** Functions are 'first class citizens'  
// ** i.e. functions are runnable objects stored in variables  
// **  
function demoFn1() {  
    console.log('executed demoFn1');  
}
```

```
console.log(demoFn1.name); // output: "demoFn1"  
demoFn1(); // output: "executed demoFn1"
```

```
const anotherVariablePointingTodemoFn1 = demoFn1  
anotherVariablePointingTodemoFn1();  
    // round brackets after a variable indicate that the function  
    // pointed to in the variable should be executed  
    // output: "executed demoFn1"
```

```
const myNotAFunctionDemoVar = "hello";  
/// next line will cause an error if uncommented  
//myNotAFunctionDemoVar(); // Error TypeError: myNotAFunctionDemoVar is not a function
```

JS-Rep 6: Higher Order Functions (HOF) take one or more Functions as Parameters

```
// **  
// ** Higher order functions (HOF) take one or more functions as parameters  
// **  
function demoHOF(fnToCall) {  
    console.log('started execution demoHOF');  
    fnToCall();  
    console.log('finished execution demoHOF');  
}  
  
demoHOF(demoFn1);  
    // output: "started ..." /n "executed demoFn1" /n "finished ..."
```

JS-Rep 7: Objects can hold Functions as Values of Properties (Methods)

```
// **  
// ** Objects can hold functions as values of properties (methods)  
// ** These can then be called as methods <obj>.method  
// **  
const reactiveObject1 = {doIt: () => console.log('doing it 1')};  
reactiveObject1.doIt(); // output: "doing it 1"  
  
const reactiveObject2 = {doIt: function() { this.handler && this.handler();}};  
console.log('CHECKPOINT 1a');  
reactiveObject2.doIt(); // no output because this.handler === undefined  
console.log('CHECKPOINT 1b');  
  
reactiveObject2.handler = () => {console.log('handling it 2')};  
reactiveObject2.doIt(); // output: "handling it 2"
```

JS-Rep 8: Do not use 'this' (Context) in Methods defined as Arrow-Functions

```
// **  
// ** Functions (methods) using "this" (context) to reference the object called  
// ** ("object before the dot") must not be arrow functions  
// **  
const reactiveObject2b = {doIt: () => this.handler && this.handler()};  
reactiveObject2b.handler = () => {console.log('handling it 2b')};  
console.log('CHECKPOINT 2a');  
reactiveObject2b.doIt();  
// no output because this.handler === undefined (this = global object)  
console.log('CHECKPOINT 2b');
```


JS-Rep 9: Functions can be Closures

```
// **  
// ** Functions (including handler functions) can be closures.  
// ** Closures reference variables from their "birth environment".  
// ** These variables are kept alive together with the referencing closures.  
// **  
function setHandler(rObj, message, step = 1) {  
  let counter = 0;  
  rObj.handler = () => {  
    counter += step;  
    console.log('CLOSURE:', counter, message);  
  };  
}  
  
setHandler(reactiveObject2, 'handling it 2a');  
  
reactiveObject2.doIt(); // output: "CLOSURE: 1 handling it 2a"  
reactiveObject2.doIt(); // output: "CLOSURE: 2 handling it 2a"
```

JS-Rep 10: Closure-Functions are Independent of each other

```
// **  
// ** Closure-functions created by the same enclosing function are independent  
// ** of each other (do not share closure-variables)  
// ** Closure variables also include arguments to enclosing functions  
// **  
const reactiveObject2c = {doIt: function() { this.handler && this.handler();}};  
setHandler(reactiveObject2c, 'handling it 2cccc', 10);  
  
reactiveObject2c.doIt(); // output: "CLOSURE: 10 handling it 2cccc"  
reactiveObject2c.doIt(); // output: "CLOSURE: 20 handling it 2cccc"  
  
reactiveObject2.doIt(); // output: "CLOSURE: 3 handling it 2a"
```

JS-Rep 11: Creating templated objects using Factory Functions (non-standard)

```
// **  
// ** Factory functions can create objects  
// **  
function createReactiveObject(name) {  
  return {  
    name,  
    handlers: [],  
    addHandler: function(handler) {this.handlers.push(handler);},  
    doIt: function() {  
      for (const handler of this.handlers) {  
        handler();  
      }  
    },  
  };  
}  
  
const reactiveObject3 = createReactiveObject('rObj3');  
reactiveObject3.addHandler(() => console.log('handler3.1'));  
reactiveObject3.addHandler(() => console.log('handler3.2'));  
reactiveObject3.doIt(); //output: "handler3.1" /n "handler3.2"
```

JS-Rep 12: Creating templated objects with Constructor functions (old-fashioned)

```
// **
// ** Constructor functions used to be the standard way to create objects in JS
// ** Constructor functions are capitalized and must be called with new
// **
function ReactiveObject(name) {
  this.name = name;
  this.handlers = [];
  this.addHandler = function(handler) {this.handlers.push(handler)};
  this.doIt = function() {
    for (const handler of this.handlers) {
      handler();
    }
  };
}

const reactiveObject4 = new ReactiveObject('rObj4');
console.log(typeof reactiveObject4); // output: object
console.log(reactiveObject4.constructor.name); // output: ReactiveObject
console.log(reactiveObject4 instanceof ReactiveObject); // output: true
reactiveObject4.addHandler(() => console.log('handler4.1'));
reactiveObject4.addHandler(() => console.log('handler4.2'));
reactiveObject4.doIt(); //output: "handler4.1" /n "handler4.2"
```

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new>

JS-Rep 13: Creating templated objects: ES6 introduced the class construct

```
// **  
// ** ES6 introduced the class construct as the new way for creating new objects in JS  
// **
```

```
class Reactor {  
  name;  
  handlers = [];  
  constructor(name) {  
    this.name = name;  
  };  
  addHandler(handler) {this.handlers.push(handler)};  
  doIt() {  
    for (const handler of this.handlers) {  
      handler();  
    }  
  };  
}
```

```
const reactorObj1 = new Reactor('reactor1');  
console.log(typeof reactorObj1); // output: object  
console.log(reactorObj1.constructor.name); // output: Reactor  
console.log(reactorObj1 instanceof Reactor); // output: true  
console.log(reactorObj1); // output: Reactor { name: 'reactor1', handlers: [] }  
reactorObj1.addHandler(() => console.log('handler5.1'));  
reactorObj1.addHandler(() => console.log('handler5.2'));  
reactorObj1.doIt(); //output: "handler5.1" /n "handler5.2"
```

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>

https://exploringjs.com/es6/ch_classes.html

JS-Rep 14:

```
// **  
// ** ES6 class definitions support definition of setters (and getters)  
// ** Created objects can react to setting (and getting) of values  
// **  
class ReactiveElement {  
  name;  
  _hidden = false;  
  handlers = [];  
  constructor(name) {  
    this.name = name;  
  };  
  set hidden(shouldBeHidden) {  
    this._hidden = shouldBeHidden;  
    console.log(`Element ${this.name} is now ${this._hidden ? 'hidden' : 'visible'}`);  
  };  
  addHandler(handler) {  
    this.handlers.push(handler);  
  };  
  doIt() {  
    for (const handler of this.handlers) {  
      handler();  
    }  
  };  
}
```

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/set>

JS-Rep 17: ES6 class definitions support definition of setters (and getters)

```
// **  
// ** ES6 class definitions support definition of setters (and getters)  
// ** Created objects can react to setting (and getting) of values  
// **  
class ReactiveElement {  
  name;  
  _hidden = false;  
  handlers = [];  
  ...  
  set hidden(shouldBeHidden) {  
    this._hidden = shouldBeHidden;  
    console.log(`Element ${this.name} is now ${this._hidden ? 'hidden' : 'visible'}`);  
  };  
  ...  
}  
  
const rElement = new ReactiveElement('BUTTON');  
  
rElement.addHandler(() => console.log('you clicked me'));  
rElement.doIt(); // output: you clicked me  
  
rElement.hidden = true; // output: Element BUTTON is now hidden  
rElement.hidden = false; // output: Element BUTTON is now visible
```

DOM GRUNDLAGEN

Problematik

■ Wie kann ich mit JavaScript eine Webseite dynamisch anpassen?

- «DHTML»: Dynamic HTML

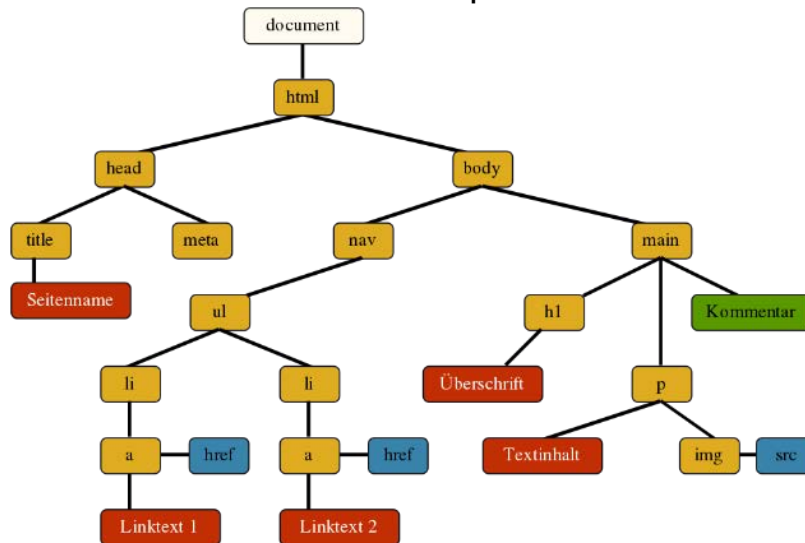
■ Was für Methoden / Funktionen sind verfügbar?

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>DOM Manipulation First Example</title>
</head>
<body>
<div id="toChange">
  Replace Me
</div>
<div id="otherDiv">
  Do not replace me
</div>
<script>
  document.getElementById('toChange').innerText = 'Replaced!';
</script>
</body>
</html>
```

01-dom-intro/03-domManipulationFirstExample.html

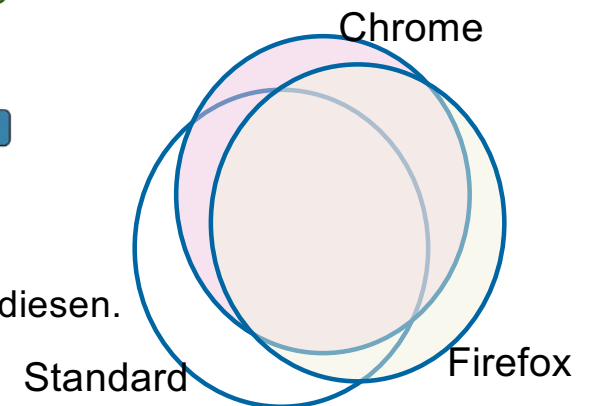
Grundlagen

- Document Object Model (DOM) ist eine Programmier-Schnittstellendefinition für Dokumente (XML/HTML)
- Das DOM repräsentiert das HTML-Dokument als Baumstruktur.
 - Jeder Node im Baum ist ein Objekt welches ein Stück vom Dokument repräsentiert
- Das DOM definiert Methoden für das Traversieren und Manipulieren des Baumes.



- **DOM Standard:**

- <https://www.w3.org/DOM/>
- <https://dom.spec.whatwg.org/>
- Hinweis: Browser implementieren den Standard unvollständig und ergänzen diesen.



Grundlagen: Tree

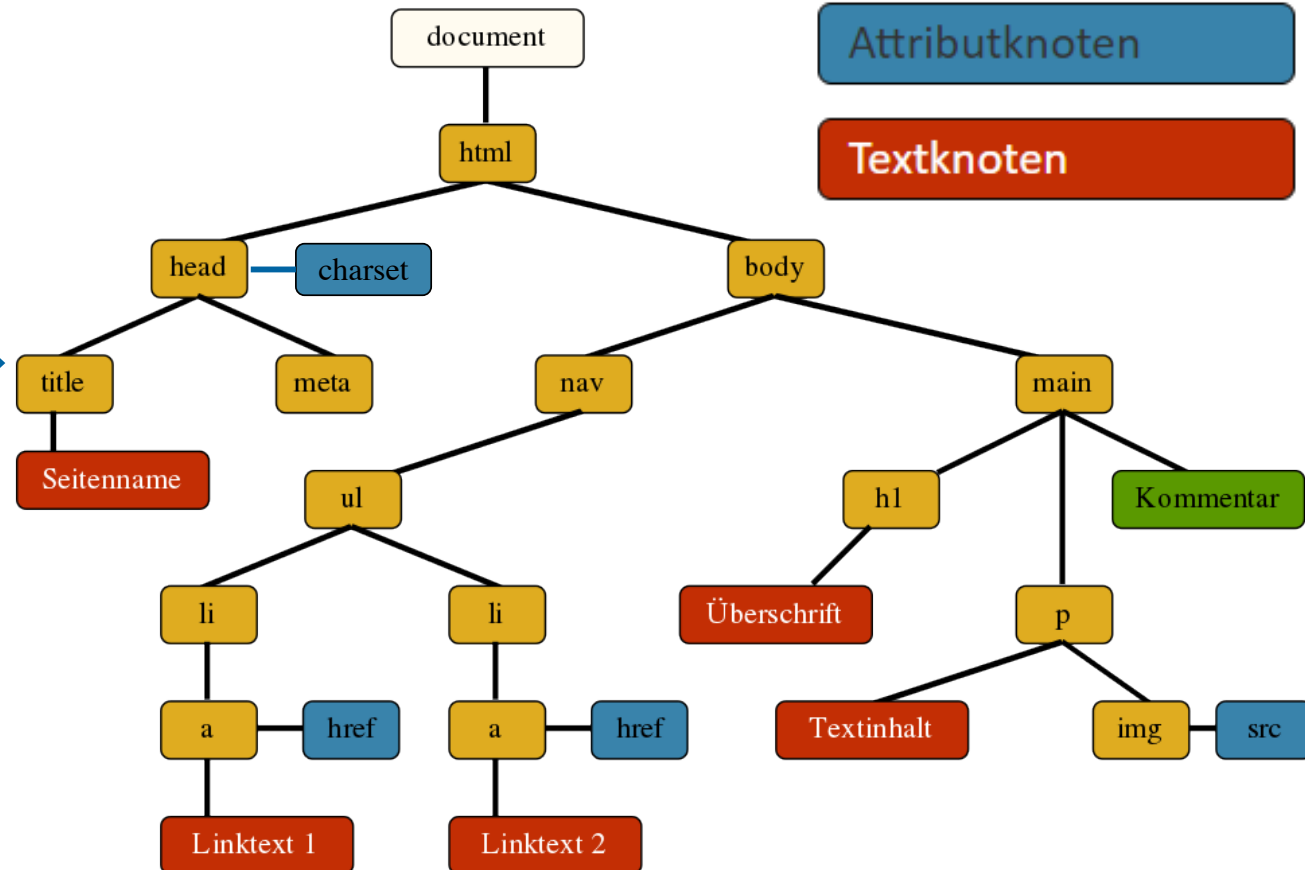
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Seitenname</title>
</head>
<body>
<nav>
  <ul>
    <li>
      <a href="#">Linktext 1</a>
    </li>
    <li>
      <a href="#">Linktext 2</a>
    </li>
  </ul>
</nav>
<main>
  <!--Kommentar-->
  <h1>Überschrift</h1>
  <p>
    Textinhalt
    
  </p>
</main>
</body>
</html>
```

01-dom-intro/04-domTreeExample.html

Elementknoten

Attributknoten

Textknoten



Attribute Nodes im DOM Baum

■ Frage: Sind Attributknoten Teil des DOM Baums?

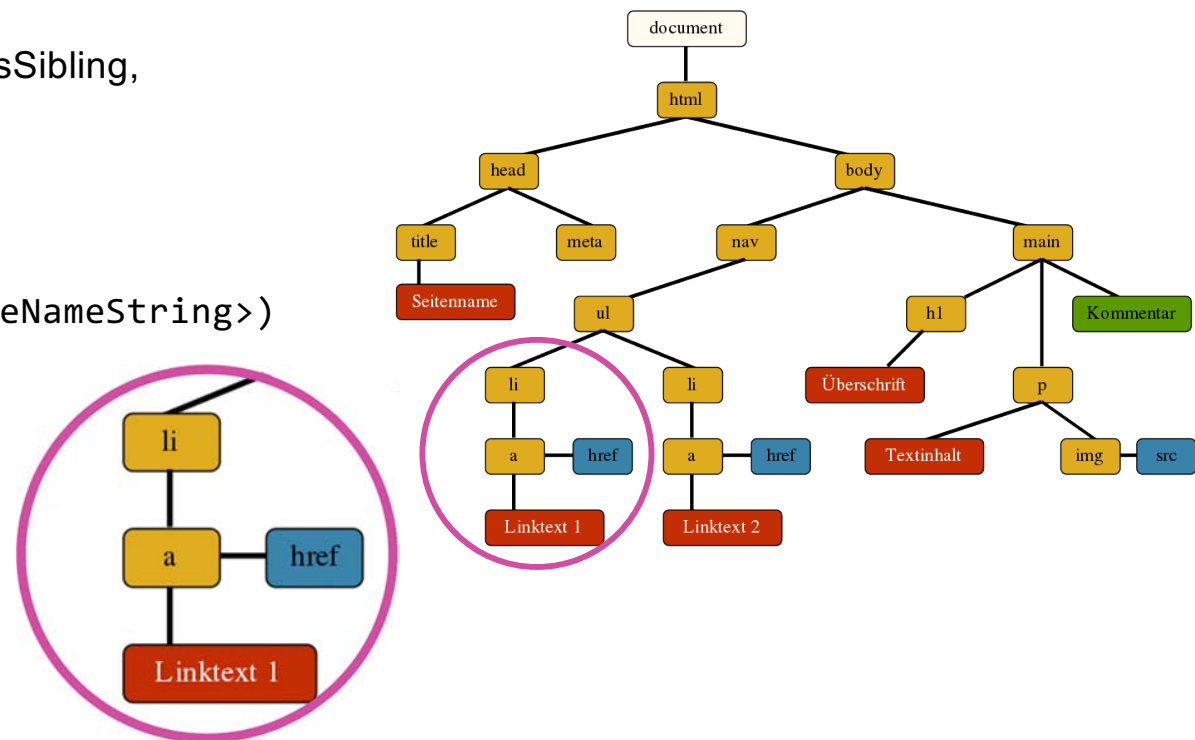
■ Antwort:

- Im Prinzip ja: Zugreifbar mit `elementNode.getAttributeNode(<attributeNameString>)`
- ABER: “not considered to be part of the DOM tree”
 - Nicht gesetzt:
`attrNode.parentNode`, `attrNode.previousSibling`,
`attrNode.nextSibling`
 - Gesetzt:
`attrNode.ownerElement`
- Daher: bevorzugt `elementNode.getAttribute(<attributeNameString>)`

Elementknoten

Attributknoten

Textknoten



<https://developer.mozilla.org/en-US/docs/Web/API/Element/getAttributeNode>

Grundlagen: DOM Tree Inspektion und Manipulation in der Chrome Dev. Konsole

Dev. Konsole

■ Elements Tab

- DOM Manipulation
- Properties Display

■ Cross-Selektion von DOM Elementen

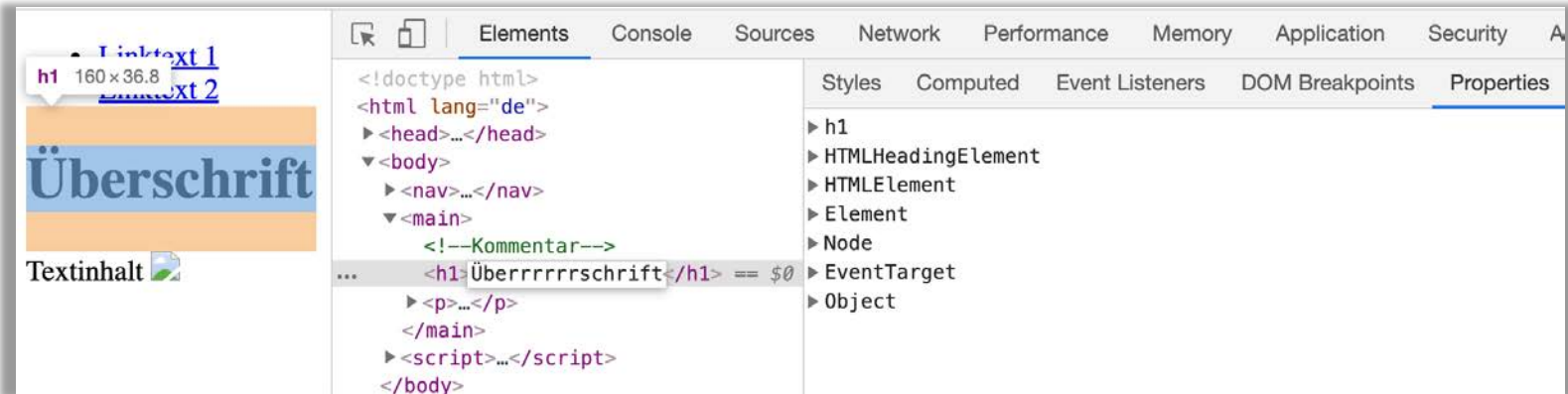
- Elements Tab <-> Display
- Console <-> Display

■ History von Selektierten Elementen

- **\$0** -> zuletzt selektiert
- **\$1** -> davor selektiert
- **\$2** ...

■ `$(<selectorString>)` = `document.querySelector()`

■ `$$(<selectorString>)` = `document.querySelectorAll()`




<https://developers.google.com/web/tools/chrome-devtools/console/utilities>

01-dom-intro/04b-domTreeExample.html

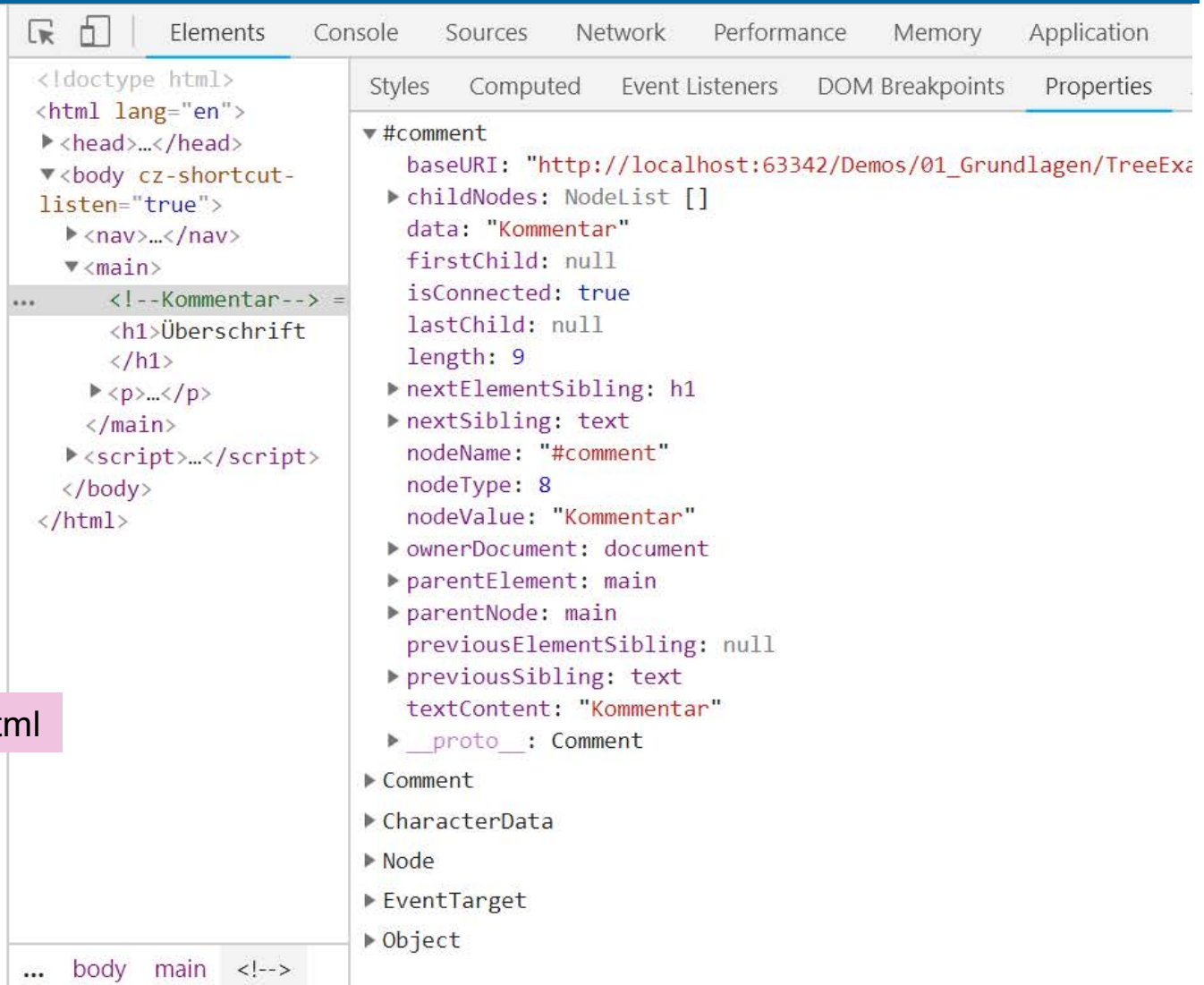
Grundlagen: DOM Tree mit DOM Nodes (unterschiedliche Typen)

- [Linktext 1](#)
- [Linktext 2](#)

Überschrift

Textinhalt 

01-dom-intro/ 04b-domTreeExample.html



The screenshot displays the browser's developer tools interface. The 'Elements' panel on the left shows the HTML document structure, with the comment node `<!--Kommentar-->` selected. The 'Properties' panel on the right shows the properties of this `#comment` node, including its `data` property set to `"Kommentar"` and its `nodeValue` property also set to `"Kommentar"`. The breadcrumb at the bottom indicates the path: `... body main <!-->`.

```
<!doctype html>
<html lang="en">
  ><head>...</head>
  ><body cz-shortcut-
listen="true">
    ><nav>...</nav>
    ><main>
      ... <!--Kommentar--> =
      <h1>Überschrift
      </h1>
      ><p>...</p>
      </main>
      ><script>...</script>
    </body>
  </html>
```

Properties of `#comment`:

- `baseURI`: "http://localhost:63342/Demos/01_Grundlagen/TreeExa"
- `childNodes`: NodeList []
- `data`: "Kommentar"
- `firstChild`: null
- `isConnected`: true
- `lastChild`: null
- `length`: 9
- `nextElementSibling`: h1
- `nextSibling`: text
- `nodeName`: "#comment"
- `nodeType`: 8
- `nodeValue`: "Kommentar"
- `ownerDocument`: document
- `parentElement`: main
- `parentNode`: main
- `previousElementSibling`: null
- `previousSibling`: text
- `textContent`: "Kommentar"
- `__proto__`: Comment

Comment

CharacterData

Node

EventTarget

Object

... body main <!-->

Wichtige globale Browser Objekte

■ window

- Globale Variablen liegen auf dem window
- Stellt alle anderen globalen Objekte zu Verfügung wie
 - console
 - history
 - document
 - ...
- <https://developer.mozilla.org/en-US/docs/Web/API/Window>

■ document

- Einstiegspunkt für den DOM-Tree
- Bietet DOM-Such-Methoden an
- Bietet DOM-Manipulations-Methoden an
- Bietet sonstige document relevanten Methoden an:
 - <https://developer.mozilla.org/en-US/docs/Web/API/Document>

DOM SUCHE / «SELEKTION»

DOM Suche / «Selektion»

■ Wie können Elemente gefunden werden?

■ Klassisch:

- `<searchRootElement>.getElementById(<idString>)`
- `<searchRootElement>.getElementsByName(<nameString>)` // for Form-Elements
- `<searchRootElement>.getElementsByClassName(<singleClassString>)`
- `<searchRootElement>.getElementsByTagName(<tagString>)`

■ «Modern»:

- `<searchRootElement>.querySelector(<selectorString>)` // first match
- `<searchRootElement>.querySelectorAll(<selectorString>)` // NodeList
- `<searchRootElement>.closest(<selectorString>)` // first matching ancestor
- `<searchRootElement>.matches(<selectorString>)` // boolean

DOM Suche: Beispiel

```
<nav>
  <ul class="nav">
    <li class="nav-item">nav1</li>
    <li class="nav-item">nav2</li>
    <li class="nav-item">nav3</li>
  </ul>
</nav>
<main>
  <button id="addButton">Hinzufügen</button>
  <ul id="list-container">
    <li>1</li><li>2</li><li>3</li><li>4</li>
  </ul>
</main>
<script>
  const listContainer = document.getElementById('list-container');
  console.log(listContainer);

  const lis = document.getElementsByTagName('li');
  console.log(lis);

  const navItems = document.getElementsByClassName('nav-item');
  console.log(navItems);

  const nav = document.getElementsByClassName('nav');
  console.log(nav);
</script></body></html>
```

▶ `<ul id="list-container">...`

▶ HTMLCollection(7)

▶ HTMLCollection(3)

▶ HTMLCollection(1)

02-dom-element-selection/
01_getElementsByXYSelection.html

DOM Suche: Beachten bei `HTMLCollection`

`getElements_By[TagName/Name/ClassName]...`

- **... geben kein Array zurück sondern eine `HTMLCollection`**
 - Auswirkung: Array Funktionen funktionieren nicht
 - Kein: `forEach`, `map`, `filter`
 - Looping möglich: `for (elmt of htmlCollection) { ... }`
- **... erzeugen ein Live-Query**
 - Beim Anpassen des DOM wird die Collection angepasst (Performance-Drain)
- **... geben immer eine Collection zurück und nicht einzelne Elemente**

02-dom-element-selection/
01_getElementsByXYSelection.html

DOM Suche: Beachten bei `NodeList` und `HTMLCollection`

■ Problem ähnlich bei `querySelectorAll()`

■ 02-dom-element-selection 02_querySelectorSelection.html

- `querySelectorAll` liefert **`NodeList`**
- **`NodeList`**: statisch, definiert `forEach` Funktion
- Kein: `map`, `filter`, ...
- Looping möglich: `for (elmt of nodeList) { ... }`

■ Für Array Funktionen:

- `HTMLCollection` / `NodeList`
zu einem Array konvertieren
`Array.from(...)`

02-dom-element-selection/ 03_HTMLCollectionNodeListIterating.html

```
for (let i = 0; i < navItems.length; i++) {  
  console.log("for", navItems[i]);  
}
```

```
for (const item of navItems) {  
  console.log("for of", item);  
}
```

```
Array.from(navItems)  
  .filter((el, i) => i % 2 === 0)  
  .forEach(el => el.style.background = "green");
```

DOM Suche: Beachten bei NodeList und HTMLCollection (cont.)

Bei einem Live-Query wird die Collection immer angepasst; wenn sich der DOM ändert.

■ Problem:

- Unerwartet
- (Performance)

■ Lösungen (falls nicht erwünscht)

- Zu einem Array konvertieren
- `querySelectorAll()` nutzen

```
<h1>The HTMLCollection Is Live</h1>
<main>
  <ul>
    <li>1st</li>
    <li>2nd</li>
    <li>3rd</li>
    <li>4th</li>
    <li>5th</li>
  </ul>
</main>
<script>
  const liElements = document.getElementsByTagName('li');

  for (let i = 0; i < liElements.length; i++) {
    liElements[i].remove();
  }
  console.log(document.querySelectorAll('ul')[0].children.length);
</script>
```

002-dom-element-selection /
4_HTMLCollectionIsLive.html

Selektion: `getElementsBy...`(...) vs. `querySelector(...)`

- `getElementsBy[TagName/Name/ClassName]` gibt immer ein **Collection** zurück.

- **Problem**

- Unsön und ineffizient wenn nur das erste Element benötigt
z.B. Element mit der Klasse `js-note-container`
 - `document.getElementsByClassName('u1')[0]`

- **Lösung:**

- `querySelector()` nutzen

querySelector / querySelectorAll etc.

■ **querySelector / querySelectorAll / closest / matches**

- <https://caniuse.com/#feat=querySelector>
- Ermöglicht es Elemente zu selektieren.
 - Syntax identisch zu den CSS-Selektoren
- Erzeugt keine Live-Queries

■ **querySelector**

- Gibt das erst im sub-Baum gefundene Element zurück

■ **querySelectorAll**

- Gibt alle im sub-Baum gefundenen Elemente zurück (NodeList)

■ **closest**

- Gibt erstes Vorfahrenelement zurück welches auf den selectorString passt

■ **matches**

- Bewertet ob das aktuelle Element dem selectorString entspricht

querySelector / querySelectorAll - Beispiel

```
const listContainer = document.querySelector('#list-container');  
console.log(listContainer);
```

```
const lis = document.querySelectorAll('li');  
console.log(lis);
```

```
const navItems = document.querySelectorAll('.nav-item');  
console.log(navItems);
```

```
const nav = document.querySelector('.nav');  
console.log(nav);
```

```
const listContainerItems = document.querySelectorAll('#list-container li');  
console.log(listContainerItems);
```


Relative Suche: Selektion nicht nur mit Start «document»

■ Die Element implementieren ebenfalls die Such-Methoden

Vorteile:

- Performanter
- lesbarer und »wiederverwendbar"

```
<nav>
  <ul class="nav">
    <li class="nav-item">nav1</li>
    <li class="nav-item">nav2</li>
    <li class="nav-item">nav3</li>
  </ul>
</nav>
<main>
  <ul id="list-container">
    <li>1st</li><li>2nd</li><li>3rd</li><li>4th</li>
  </ul>
</main>
<script>
```

02-dom-element-selection/
05_subtree_selection.html

```
console.log("1", document.getElementsByClassName('nav')[0].getElementsByTagName("li").length);
```

```
console.log("2", document.querySelector('.nav').querySelectorAll("li").length);
```

```
</script>
```

closest - Beispiel

```
const liElements = document.querySelectorAll('li');
console.log(liElements);

function isInNav(element) {
  const maybeNav = element.closest('nav');
  return Boolean(maybeNav);
}

function inNavText(element) {
  return `is ${isInNav(element) ? '' : 'not '}in the Nav Area`;
}

console.log('first li-Element',
  inNavText(liElements[0]));
//output: first li-Element is in the Nav Area

console.log('last li-Element',
  inNavText(liElements[liElements.length - 1]));
// output: last li-Element is not in the Nav Area
```

```
<nav>
  <ul class="nav">
    <li class="nav-item">nav1</li>
    <li class="nav-item">nav2</li>
    <li class="nav-item">nav3</li>
  </ul>
</nav>
<main>
  <ul id="list-container">
    <li>1st</li>
    <li>2nd</li>
    <li>3rd</li>
  </ul>
</main>
```

02-dom-element-selection/
06_closest-demo.html

match - Beispiel

```
function isEvenOfType(element) {  
    return element.matches(':nth-of-type(2)');  
}  
  
function repeatTextOfALLElementsOfEvenType(elementList) {  
    for (element of elementList) {  
        if (isEvenOfType(element)) {  
            element.innerText = element.innerText + element.innerText;  
        }  
    }  
}  
  
repeatTextOfALLElementsOfEvenType(LiElements);
```

```
<nav>  
  <ul class="nav">  
    <li class="nav-item">nav1</li>  
    <li class="nav-item">nav2</li>  
    <li class="nav-item">nav3</li>  
  </ul>  
</nav>  
<main>  
  <ul id="list-container">  
    <li>1st</li>  
    <li>2nd</li>  
    <li>3rd</li>  
  </ul>  
</main>
```

DOM NAVIGATION

DOM Navigation: childNodes vs. children / firstChild vs. firstElementChild

03-dom-Navigation
domNavigationDemo.html

■ Das Node Interface stellen Methoden zu Navigation im DOM Baum bereit

häufiger verwendet sind

- `<node>.parentElement`
- `<node>.childNodes` (ACHTUNG: auch text-Nodes, und Kommentare)
- `<node>.children` -> nur die Nodes vom Typ `HTMLElement`
- `<node>.firstChild` -> erste Node: auch text-Nodes, und Kommentare
- `<node>.firstElementChild` -> erstes `HTMLElement`
- `<node>.nextSibling` -> nächstes Geschwister-NODE
- `<node>.nextElementSibling` -> nächstes Geschwister-Element

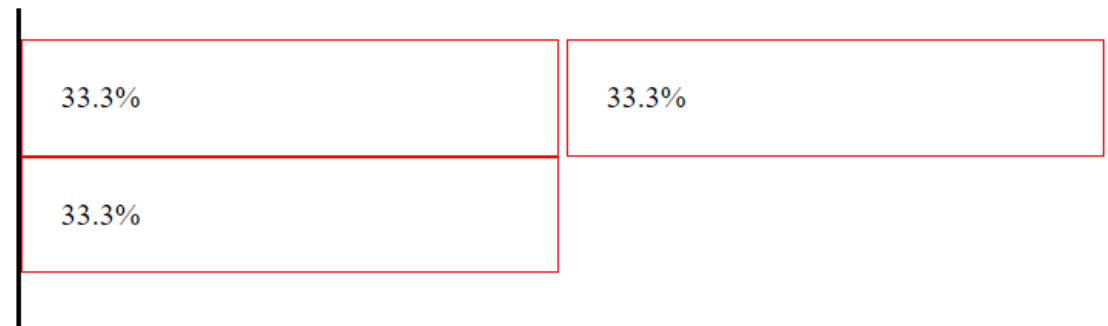
EINSCHUB: WHITESPACES

Whitespaces

```
.type1 {  
  width: calc(100% / 3);  
  display: inline-block;  
}  
  
<div id="div1" class="container">  
  <span class="type1">33.3%</span>  
  <span class="type1">33.3%</span>  
  <span class="type1">33.3%</span>  
</div>
```

```
div1  
▼ NodeList(7) [text, spa  
  length: 7  
  ► 0: text  
  ► 1: span.type1  
  ► 2: text  
  ► 3: span.type1  
  ► 4: text  
  ► 5: span.type1  
  ► 6: text  
  ► __proto__: NodeList
```

04-whitespace



- Problematik: Die Zeilenumbrüche werden als leerer Textnode im DOM-Tree repräsentiert.

Whitespace: Lösungen

1. Die Umbrüche entfernen d.h. alles auf eine Zeile
2. Die Umbrüche als Kommentare
3. ... (Pre-Processing im Build, entfernen mit JS) ...

CSS Flexbox nutzen (bekannt!)

```
<div class="container">  
  <span class="type1">33.3%</span><span class="type1">33.3%</span><span class="type1">33.3%</span>  
</div>
```

```
<div class="container">  
  <span class="type1">33.3%</span><!--  
  --><span class="type1">33.3%</span><!--  
  --><span class="type1">33.3%</span>  
</div>
```

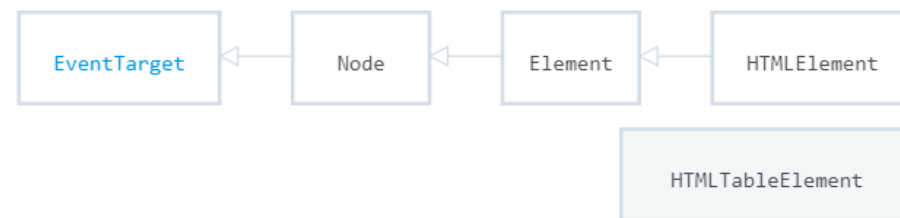

DOM MANIPULATION

DOM Manipulation

■ Knoten im DOM Tree (Table, Liste, Document, ...) implementiert unterschiedliche Interfaces

- Beispiel HTMLTableElement: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableElement>

- HTMLElement
- Element
- Node
- EventTarget

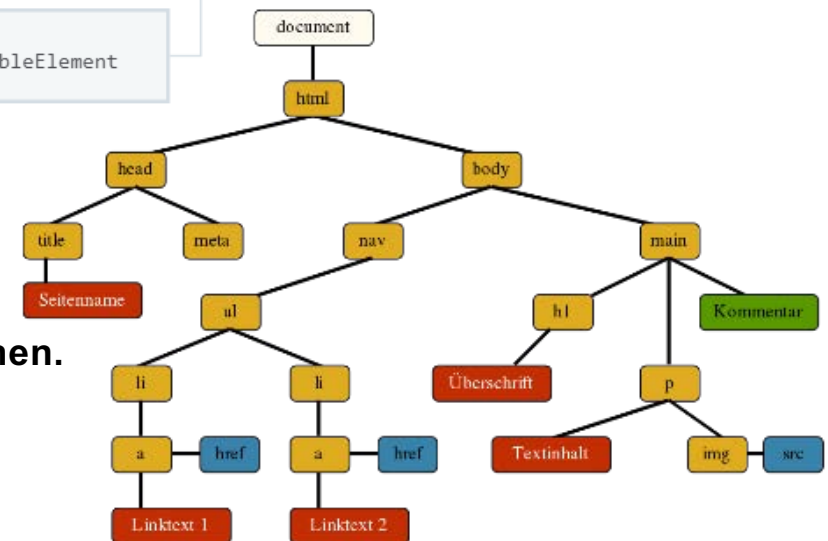


■ Jeder Knoten im DOM ist ein Node.

- Auch das document

■ Die Interfaces definieren, welche Methoden zu Verfügung stehen.

- Beispiel HTMLTableElement :
 - createCaption()
 - createTFoot()
 - createTHead()
 - ...



NodeTypen

■ EventTarget

- Diese Objekte können Events empfang und/oder senden

■ Node

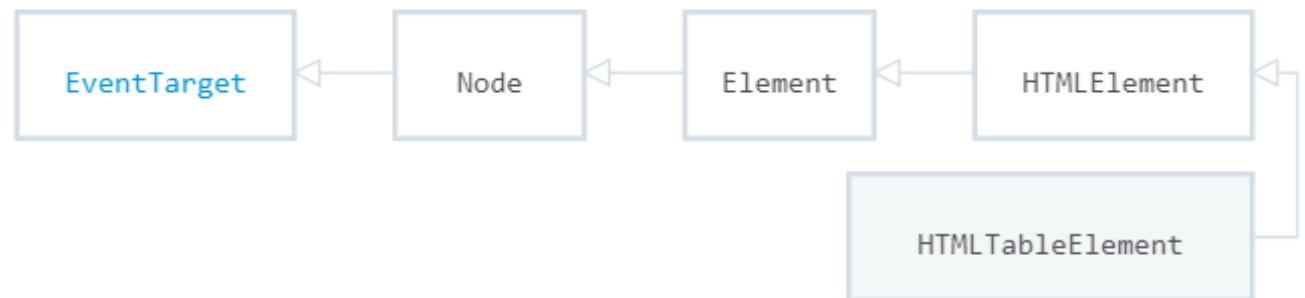
- Basis Interface damit es eine einheitliche Schnittstelle für den Tree gibt
 - Auffinden
 - Traversierung

■ Element

- Basis für alle Elemente, welche im Tree platziert werden.
- Definierte Methoden auf Stufe Element
 - Z.B. .children beinhaltet nur Elemente und nicht Nodes

■ HTMLElement

- Definiert HTML-Attribute
- Definiert weitere Events



EventTarget

- <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget>
- **Wichtige Methoden**
 - `addEventListener()`
 - `removeEventListener()`

Node Interface

<https://developer.mozilla.org/en-US/docs/Web/API/Node>

■ Wichtige Properties

- `childNodes`
- `firstChild`
- `nextSibling`

■ Wichtige Methoden

- `appendChild()`
- `removeChild()`

Element Interface

<https://developer.mozilla.org/en-US/docs/Web/API/Element>

■ Wichtige Properties

- `id`
- `className` / `classList`
- `innerHTML`
- `attributes`

■ Wichtige Methoden

- `getAttribute()` / `setAttribute()` / `toggleAttribute()`
- `closest()` : <https://developer.mozilla.org/en-US/docs/Web/API/Element/closest>
- `querySelector()` / `querySelectorAll()` / ...
- `insertAdjacentHTML()` / ...
- `scrollTo()`

Element Interface: ParentNode

<https://developer.mozilla.org/en-US/docs/Web/API/ParentNode>

<https://developer.mozilla.org/en-US/docs/Web/API/ChildNode>

■ Das Element implementiert auch das ParentNode und ChildNode Interface

- children
- firstElementChild
- lastElementChild

■ Wichtige Methoden

- append()
- remove()

HTMLElement Interface

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement>

■ Wichtige Properties

- dataset
- style
- hidden

Neuen Node / Element erstellen

- **document.createElement(<tagName>)**
 - Erzeugt ein Element mit dem tagName
- **document.createTextNode(<content>)**
 - Erzeugt ein Text-Node mit dem Inhalt
- **document.createDocumentFragment()**
 - Erzeugt ein Node welcher beim Anhängen an einen Parent (z.B. appendChild) entfernt wird.
 - Statt dem Fragment-Knoten selber werden dessen Kinder dem neuen Parent hinzugefügt.
- **document.create[...] ()**

Neues Element erstellen (document.createElement, document.appendChild, ...)

05-dom-manipulation/
01_DomCreate.html

```
const newEL = document.createElement('div');  
newEL.appendChild(document.createTextNode('Hello'));  
document.getElementById("container").appendChild(newEL);  
  
const newEL2 = document.createElement('div');  
newEL2.innerText = 'World';  
document.getElementById("container").appendChild(newEL2)
```

createElement Demo

Hello
World

Neues Element erstellen (insertAdjacentHTML)

05-dom-manipulation/
01b_DomCreate2.html

```
const container = document.getElementById("container");  
container.insertAdjacentHTML('beforebegin', '<div>beforebegin</div>');  
container.insertAdjacentHTML('afterbegin', '<div>afterbegin</div>');  
container.insertAdjacentHTML('beforeend', '<div>beforeend</div>');  
container.insertAdjacentHTML('afterend', '<div>afterend</div>');
```

insertAdjacentHTML Demo

beforebegin
afterbegin
beforeend
afterend

Weitere DOM-Manipulations-Funktionen

■ **appendChild**

- <https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild>

■ **insertBefore**

- <https://developer.mozilla.org/en-US/docs/Web/API/Node/insertBefore>

■ **removeChild**

- <https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild>

■ **replaceChild**

- <https://developer.mozilla.org/en-US/docs/Web/API/Node/replaceChild>

■ **insertAdjacentHTML**

- <https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML>

■ **insertAdjacentElement**

- <https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentElement>

innerHTML

■ Liest oder schreibt den Inhalt vom Element als HTML Code

- <https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML>

```
const c = document.getElementById("container");  
console.log(c.innerHTML);  
c.innerHTML = '<div>Changed</div>';  
console.log(c.innerHTML);
```

```
c.innerHTML = '';
```

■ Was passiert?

```
c.innerHTML = '<script>alert('Hi')</script>';  
c.innerHTML = '<img src='x' onerror='alert("hacked")'>';
```

**Gefahr:
Cross Site Scripting**

textContent, innerText

05-dom-manipulation/
04_innerText.html

■ Node.textContent

- Beinhaltet den kompletten Text vom Element.

■ HTMLElement.innerText

- Beinhaltet nur **sichtbaren** Text vom Element (bedeutet Reflow vor Auslesen)

■ textContent / innerText

- Ersetzt den Inhalt durch einen neuen Text mit «Escaping» -> '>' -> >
- XSS verhindert (aber auch Formatierung)

```
<div id="container">
  blablaba
  <span>Hello</span>
  <span style="display: none">World</span>
</div>
<script>
  const c = document.getElementById("container");
  console.log(c.innerText);
  console.log(c.textContent);
  c.innerText = "changed";
  c.textContent = "changed";
</script>
```

<https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent>

<element>.classList vs. className

05-dom-manipulation/
05_classNameVsClassList.html

■ <element>.className

- Die alte Variante um mit CSS Klassen umzugehen.
- Die Klassen werden als String repräsentiert, getrennt mit einem « »

■ <element>.classList

- Die Klassen werden in einer Liste (Eine DOMTokenList) aufbereitet. DOMTokenList bietet nützliche Hilfsfunktionen:
add / remove / toggle / contains / replace
- classList ist zu bevorzugen.

```
<div id="e1" class="box alert important"></div>
```

```
<script>
```

```
    console.log(document.getElementById("e1").className);  
    // box alert important
```

```
    console.log(document.getElementById("e1").classList);  
    // DOMTokenList(3) ["box", "alert", "important"]
```

```
</script>
```

<element>.style

05-dom-manipulation/
06_styleManipulation.html

- Das Property `style` eines Elements ist vom Typ `CSSStyleDeclaration`
- Die Inline Styles (CSS Eigenschaften) des Elements können als Properties der `CSSStyleDeclaration` gesetzt werden
 - CSS Eigenschaften werden statt Kebap-Case im CamelCase geschrieben
Also: `fontSize` statt `font-size`
 - Abfrage der vollständigen aktuellen Eigenschaften `window.getComputedStyle(<element>)`
 - Besser als direktes Setzen des `style` Attributs als String

```
<body>
<h1>Style Manipulation Demo</h1>
<div id="theDiv">Style Me</div>
<script>
  const theDiv = document.getElementById("theDiv");
  console.log(theDiv.style);
  console.log(window.getComputedStyle(theDiv));
  theDiv.style.background = "yellow";
  theDiv.style.fontSize = "30px";
  console.log(theDiv.style);
</script>
```


DOM Manipulation Performance (im 2020)

05-dom-manipulation
01c_DomCreatePerformance.html

- Schnell (Chrome & Firefox); Nutzung von `Fragment` ohne Unterschied (auch gut: Zuerst HTML-String konstruieren mit `+=`, dann einmalige Zuweisung zu `innerHTML`)

```
function appendLiTextContent(parentListElement, textContent) {  
    const newLi = document.createElement('li');  
    newLi.textContent = textContent;  
    parentListElement.appendChild(newLi);  
}
```

- Etwas langsamer (x 5) in Chrome

```
function appendLiAdjacentHTML(parentListElement, textContent) {  
    parentListElement.insertAdjacentHTML('beforeend', `- ${textContent}</li>`);  
}

```

- Multiple Zuweisungen von `innerHTML` = VIEL langsamer (x 5000) in Chrome und Firefox
-> **DON'T DO THIS**

```
ulElement.innerHTML += `- ${String(i)}</li>`;

```

ATTRIBUTE

Attribute vs. DOM-Element Properties

- Im HTML werden Attribute definiert:

```
<input type="button" value="value-content"  
      class="info-class" style="background: aqua" required/>
```

- Diese werden auf die Properties von HTMLElement und Unterklassen (und auch Element) API abgebildet

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement>

- Beispiel: <HTMLButtonElement>.disabled

- Die Namen von Properties können sich von denen der Attribute unterschiedlich sein, z.B.

- class => «className» bzw. «classList»

- for => htmlFor

```
const iElmt = document.querySelector("input");
```

```
console.log(iElmt.type, iElmt.value, iElmt.className, iElmt.style);
```

Setzen Attribute vs. Setzen von Properties

06_attributes/
01_prop_vs_attr.html

■ Viele Attribute können über Properties gesetzt werden ABER

- Nicht alle Properties sind gemapped, aber die meisten.
- Nicht alle Zuweisungen von Properties aktualisieren das entsprechende Attribut
- Nicht-Standard-Attribute können nicht über das Property gesetzt werden.
 - aria-* Attribute
 - data-* Attribute

■ `setAttribute`

- Setzt beliebige Werte beliebiger Attribute
- Setzt auch das entsprechende Property (meist)

■ `removeAttribute`

- Löscht das Attribut (wichtig für boolsche Attribute)

```
<input id="inptBtn1" type="button" value="btnInitValue">
<input id="inptTxt1" value="txtInitValue">
<script>
  const inputBtn1 = document.querySelector('#inptBtn1');
  const inputTxt1 = document.querySelector('#inptTxt1');
  inputBtn1.value = 'newInputBtnPropValue'; // causes attribute update
  inputTxt1.value = 'newInputTxtPropValue'; // no attribute update
```

Attribute-Typen

■ Boolesche Attribute (HTML) -> bekannt

- Sind «true», falls definiert.
- Falls nicht definiert: «false»

`<input type="checkbox" checked>`

`<input type="checkbox" checked="false">` // nicht möglich; muss entfernt werden

- Hinweis: Via Propertyzuweisung kann das boolesche Attribute geändert werden.
Problem: Zuweisung von `false` löscht das Attribute nicht aus dem DOM.
CSS `:checked` (Pseudoklasse-Selektor) funktioniert,
CSS `input:checked` (Attribut-Selektor) funktioniert nicht.
=> Property Manipulation und CSS Attribute Selectors nicht zusammen nutzen.

```
#cbx1[checked] {
  /*problem*/
  outline-offset: 2px;
  outline: #EE001C 3px solid #EE001C;
}
#cbx2:checked {
  /*works*/
  outline-offset: 2px;
  outline: #EE001C 3px solid #EE001C;
}
```

■ Alle anderen Attribute: Der Value ist immer ein «String»

- Der W3C Validator kennt den gewünschten Type z.B. `step` erwartet eine Zahl.
Dies wird aber nicht enforced.
- Im Property wird der Attribute-Wert immer als String repräsentiert.



06_attributes/
03_checkBoxCheckedDemo.html

06_attributes/
02_typen_remove.html

DOM Handling = JQuery ?!?



DOM Handling = JQuery !!

Web Almanac:

Analyse des httparchive July 2019 crawl:

5,790,700 websites in dataset. Among those,
5,297,442 mobile websites, 4,371,973 desktop websites

<https://almanac.httparchive.org/en/2019/javascript>

jQuery, the most popular JavaScript library ever created, is **used in 85.03% of desktop pages** and **83.46% of mobile pages**. The advent of many Browser APIs and methods, such as Fetch and querySelector, standardized much of the functionality provided by the library into a native form.

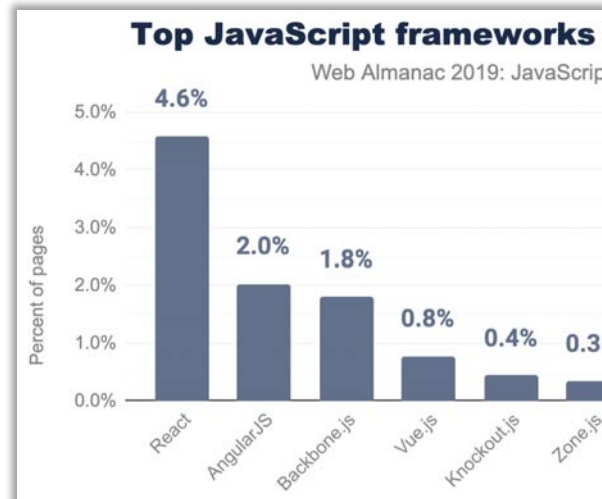
Q: Although the popularity of jQuery may seem to be declining, why is it still used in the vast majority of the web?

A:

- Wordpress (30% sites)
- **Legacy**



<i>Library</i>	<i>Desktop</i>	<i>Mobile</i>
jQuery	85.03%	83.46%
jQuery Migrate	31.26%	31.68%
jQuery UI	23.60%	21.75%
Modernizr	17.80%	16.76%
Fancybox	7.04%	6.61%



JQuery



Smashing Magazine ✓
@smashingmag

How often do you use jQuery in your current or ongoing projects these days?



1,824 votes · 5 days left

9:57 AM · May 18, 2020 · [Twitter Web App](#)

<https://twitter.com/smashingmag/status/1262291309169082368>



Ryzal Yusoff 🇬🇧🇲🇾🇮🇩 @ryzalyusoff · 8h

Replying to @smashingmag

That's all I use.



Connor Bode @connorbode · 4h

Replying to @smashingmag

I usually need to install it as a dependency. For example, i use datatables.net all the time and that relies on jQuery.



Shane Hudson @ShaneHudson · 10h

Replying to @smashingmag

All the time for legacy reasons. In a new project? Never. I can't think of any reason to use these days.



Mike Rogers 🇬🇧 @MikeRogers0 · 10h

Same! I haven't had to touch it to much lately (newer projects using Stimulus), but it does pop up quite often in a few older projects. I don't have any overwhelming desire to remove it urgently from older stuff, for the most part I can just leave it ticking over.



Roberto Serra @_obiTheOne · 7h

Replying to @smashingmag

way too often... In a modern project, there would be no need, but when you work on a project bootstrapped by someone who lives in 5yrs ago JS era...

You don't need JQuery Any More

- <https://github.com/nefe/You-Dont-Need-jQuery>
- <https://hackernoon.com/you-truly-dont-need-jquery-5f2132b32dd1>
- <https://dev.to/kdinnypaul/you-don-t-need-jquery-1a18>
- <https://dev.to/adnanbabakan/i-don-t-need-jquery-anymore-so-don-t-you-perhaps-3nj>

```
function $(queryString) {  
  return document.querySelector(queryString);  
}
```

```
function $$ (queryString) {  
  return document.querySelectorAll(queryString);  
}
```

```
console.log($('button')); //-> <button>b1</button>  
$('button').classList.add('dangerous'); // works  
$$('button').classList.add('dangerous'); // does not "work"
```



You don't need JQuery Any More (2)

12-Jquery/SansJQuery.html

Statt implizite (versteckte) Iteration
Empfohlen: Explizites Iterieren

```
//$$('button').classList.add('dangerous'); // does not "work"
```

```
//  
// best way of making it work -> explicit iteration  
//  
for (button of $$('button')) {  
    button.classList.add('dangerous');  
}
```

You don't need JQuery Any More (3)

12-Jquery/SansJQuery.html

(Schlechtere) Alternative: Selber JQuery-ähnliches API bereit stellen

```
function $$ (queryString) {  
  const elements = document.querySelectorAll(queryString);  
  elements.classList = {  
    add: function(classesStr) {  
      elements.forEach(e => e.classList.add(classesStr));  
    },  
  };  
  return elements;  
}
```

```
$$('button').classList.add('dangerous'); // "works" (no further chaining)
```

```
// Optional exercise: recreate JQuery API including chaining for  
// $("button").addClass(), ...removeClass(), ...data(),  
// Solution: 12-JQuery/SansJQueryExerciseSolution1-4.html (-> OO JS!)  
//  
// also see https://github.com/nuxodin/domProxy (
```

ÜBUNG

Übung: innerHTML / appendChild

- Die Songs in der Liste sollen mit JavaScript dem DOM hinzugefügt werden
- Auf 2 verschiedene Varianten:
 - innerHTML
 - Die Liste soll komplett als Template-String definiert werden
 - createElement(), appendChild()
 - Die Liste soll mithilfe der DOM-Methoden erstellt werden.
- Vorlage: 99_Uebungen/Vorlagen
 - 01_InnerHTML.html und 02_AppendChild.html
 - songmodel.js: Beinhaltet die Hilfsfunktionen und die Daten.
- Lösung: 99_Uebungen/solution

Songs

- **Thank you for the music**

ABBA

- **California Girls**

Beach Boys

- **How Deep Is Your Love**

Bee Gees

DOM EVENTS

Events

■ Events ermöglichen auf Ereignisse zu reagieren. Beispiele:

- Button wird betätigt.
- Fenster wird verkleinert.
- Der Mauszeiger verlässt einen gewissen Bereich.
- Video wird abgespielt.
- Eine Taste wurde geklickt.
- Die Webseite hat keine Internet-Verbindung mehr.
- ...

■ API:

- <https://developer.mozilla.org/en-US/docs/Web/Events>
- <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget>

■ Zum nachlesen:

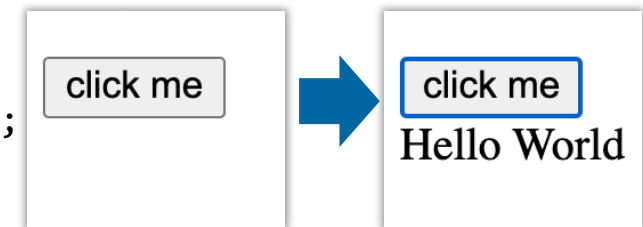
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

Events: Erstes Beispiel

```
<button id="btnClickMe">click me</button>
<div id="container"></div>
```

```
<script>
  const btn = document.querySelector("#btnClickMe");
  const container = document.querySelector("#container");
  const eventListener = () => {
    container.appendChild(document.createTextNode("Hello World"));
  };
  btn.addEventListener("click", eventListener);
</script>
```

07-event-handling/
01_DOMEventsDemo1.html



- Jedes HTML-Element bietet diverse Events an auf welche EventListener registriert werden können. Mit `addEventListener` wird ein oder mehr EventListener beim Element registriert (auch «subscribe», «listen», «bind»)
- Der EventListener (auch EventHandler, Delegate, Callback, ...) ist die Funktion, welche beim Eintreten des Events aufgerufen wird. («invoke», «fire», «trigger»)



Registration von EventListeners

Es gibt 3 Möglichkeiten um Events zu registrieren

1. addEventListener

- Die *empfohlene* Variante
- Mehrere registrierte Listener für das gleiche Event möglich
- Wird nicht beeinflusst durch die anderen Arten der Registrierung

2. Property

- Bei Property und Inline haben alle Events ein «on»-Prefix.
- Jede Zuweisung überschreibt vorherige Zuweisungen.
- Überschreibt Inline Registrierung. Letzte Zuweisung «gewinnt». Nur eine Listener Funktion möglich

3. Inline

- Für Prototypen / Generierten Code / Prüfung (/ Single Page Frameworks) geeignet.

```
<button id="1">1</button>
<button id="2">2</button>
<button onclick="alert('3')">3</button>
<script>
  document.getElementById("1").addEventListener("click", () => alert('1'));
  document.getElementById("2").onclick = () => alert('2');
</script>
```

07-event-handling/02_eventListenerRegistrationOptions.html

(1) addEventListener
-> "empfohlen"

(2) Zuweisung "on..." Property -> nicht empfohlen

(3) Inline Handler "on..."
-> nicht empfohlen

addEventListener

■ **target.addEventListener(type, listener[, options]);**

07-event-handling/
03_addEventListener.html

- type
 - Name vom Event: <https://developer.mozilla.org/en-US/docs/Web/Events>
- listener,
 - Die EventListener Funktion
 - Inline oder separat definiert
- Options Objekt
 - capture: t/f - Reagiert auf Events in der Capture-Phase
 - once: t/f - Der Listener wird nach der ersten Aktivierung entfernt.
 - passive: t/f - Für Performance-Optimierungen

■ **target.addEventListener(type, listener[, useCapture]);**

- Alte Variante, erlaubt nur «capture»

Event-Phasen

■ Events durchlaufen 3 Phasen

1. Capture-Phase

- Das Event durchläuft den DOM Tree vom Root zum Leaf.
- Jedes Element kann reagieren

2. Target Phase

- Das Event wird auf dem «Ziel» ausgelöst.

3. Bubble Phase

- Das Event durchläuft den DOM Tree vom Leaf zum Root.
- Nicht jedes Event durchläuft die Bubble-Phase:
Bubble: <https://developer.mozilla.org/en-US/docs/Web/Events/click>
Kein Bubble: <https://developer.mozilla.org/en-US/docs/Web/Events/>

07-event-handling/
05_eventPhase.html

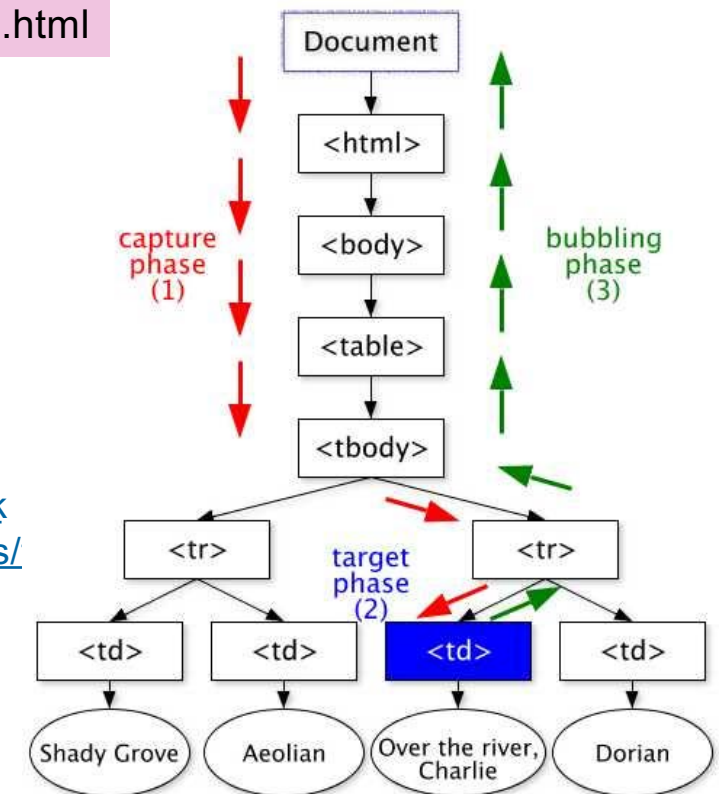


Bild: <https://www.w3.org/TR/2003/NOTE-DOM-Level-3-Events-20031107/events.html#Events-phases>

Event-Objekt

- Meistens reicht es nicht aus zu wissen, ob ein Event ausgelöst wurde.
- Die meisten Events bieten Zusatzinformationen an.
 - Event: «change» - Was wurde geändert?
 - Event: «keydown» – Welche Taste wurde gedrückt?
- Lösung: Der EventListener erhält diese Informationen als Argument (definiert Parameter)
- Beispiel: <https://developer.mozilla.org/en-US/docs/Web/Events/keydown>

07-event-handling/
04_event_Args.html

```
<body>
  <input>

  <script>
    document.querySelector("input").addEventListener("keydown", (event) => {
      console.log(event.key);
    })
  </script>
```

Event-Objekt

- <https://developer.mozilla.org/en-US/docs/Web/API/Event>
- **Das Event Objekt erbt vom generischen Event-Interface.**
Für die Events gibt es spezifische Typen z.B.
 - MouseEvent
 - WheelEvent
 - InputEvent
- **Das Basis-Interface bietet die folgenden wichtigen Properties / Methoden an**
 - target
 - currentTarget
 - stopPropagation()
 - preventDefault()

Event-Properties

- **target**
 - Zeigt auf das Target-Element vom Event
- **currentTarget**
 - Zeigt auf das aktuelle Element d.h. das Element welches den Event-Listener registriert hat.
- **stopPropagation()**
 - Ermöglicht es ein Event vom «Bubbling» und "Capturing" abzuhalten.
- **preventDefault()**
 - Verhindert Default-Aktionen. Z.B. den automatischen Form-Submit

Ergänzung: Inline-Event

- Bei Inline-Events wird das Event-Objekt auf die Variable "event" gelegt.
- Im Eventhandler (unabhängig von der Registrierung) wird das Current-Target auf den Context (this) gelegt.

```
<body>  
  <input onkeydown="console.log(event.key)">  
  <input onkeydown="console.log(this.value)">  
</body>
```

Event Tracing und Debugging mit der Chrome Dev. Konsole

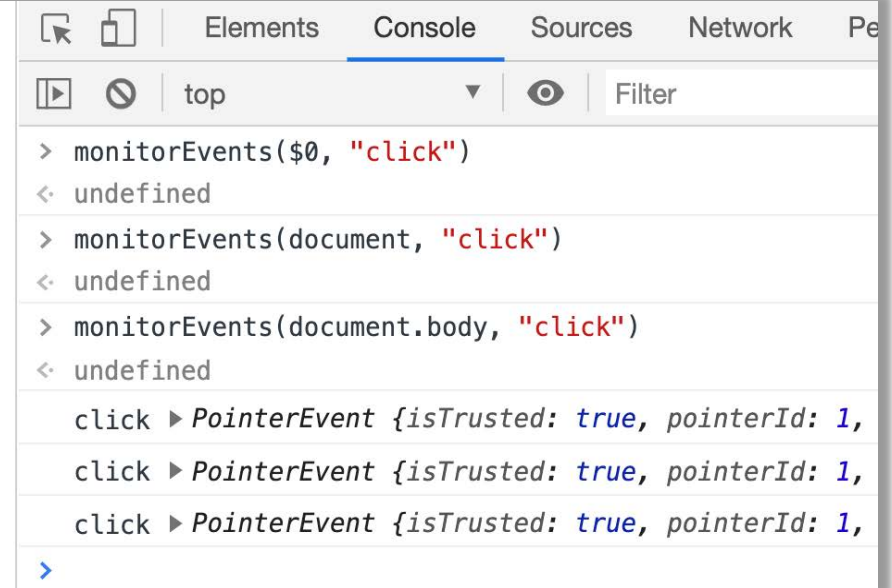
Dev. Konsole API

- `$_` Resultat der letzten Evaluation
- `inspect(<object> | <function>)`
z.B. `inspect($1)`
-> Objekt / Funktion im entsprechenden Panel anzeigen
- `getEventListeners(<object>)`
-> alle EventListeners des Objektes auflisten
- `monitor(<fn>)`
-> Tracing der Funktion mit Argumenten
- `monitorEvents(<object> [, <events>])`
z.B. `monitorEvents(window, "resize")`
-> Tracing der Eventbehandlung
- `debug(<fn>)`
-> Debugger starten bei Funktionsaufruf

DOM Events Demo 1

click me

Hello World Hello World
Hello World Hello World
Hello World Hello World
Hello World



07-event-handling/
01_DOMEventsDemo1.html

<https://developers.google.com/web/tools/chrome-devtools/console/utilities>

ÜBUNG

Inline Events

- Es soll möglich sein die Songs zu bewerten.
- Die Songs sollen nach ihrem Rating sortiert gerendert werden.
- Registrieren Sie den Eventhandler als Inline-Event
- **Challenge (!) Aufgabe:**
Die Veränderung im Rating soll animiert dargestellt werden.
- **Vorlage: 99_Uebungen/Vorlagen/03_InlineEvents.html**
 - songmodel.js: Beinhaltet die Hilfsfunktionen und die Daten.
- **Lösung: 99_Uebungen/solution**

Songs

- 3 **Thank you for the music**
ABBA
- 2 **California Girls**
Beach Boys
- 1 **How Deep Is Your Love**

DOM LIFECYCLE

Motivation

Weshalb funktioniert dieser Click-Handler nicht?

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script>
    document.getElementById("btn").addEventListener("click", () => console.log("Hello World"))
  </script>
</head>
<body>
<button id="btn">Click me</button>

</body>
</html>
```

Lifecycle

■ Das DOM kennt 3 unterschiedliche Zustände

<https://developer.mozilla.org/en-US/docs/Web/API/Document/readyState>

■ loading

- Das Dokument wird geladen

■ interactive

- Das Dokument wurde geladen aber gewisse Ressourcen sind noch nicht geladen:
 - Bilder / Video
 - Stylesheets
 - Frames
- Wird durch das Event «DOMContentLoaded» repräsentiert

■ complete

- Das Dokument und alle Ressourcen sind geladen
- Wird durch das Event «load» repräsentiert

Lifecycle

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>DOM Lifecycle Demo</title>
  <script>

    console.log(document.readyState); // Loading

    window.addEventListener("load", () => console.log(document.readyState)); // complete
    document.addEventListener("DOMContentLoaded", () => console.log(document.readyState)); // interactive

  </script>
</head>
<body>

  <script>
    console.log(document.readyState) // Loading
  </script>
</body>
</html>
```

Lifecycle

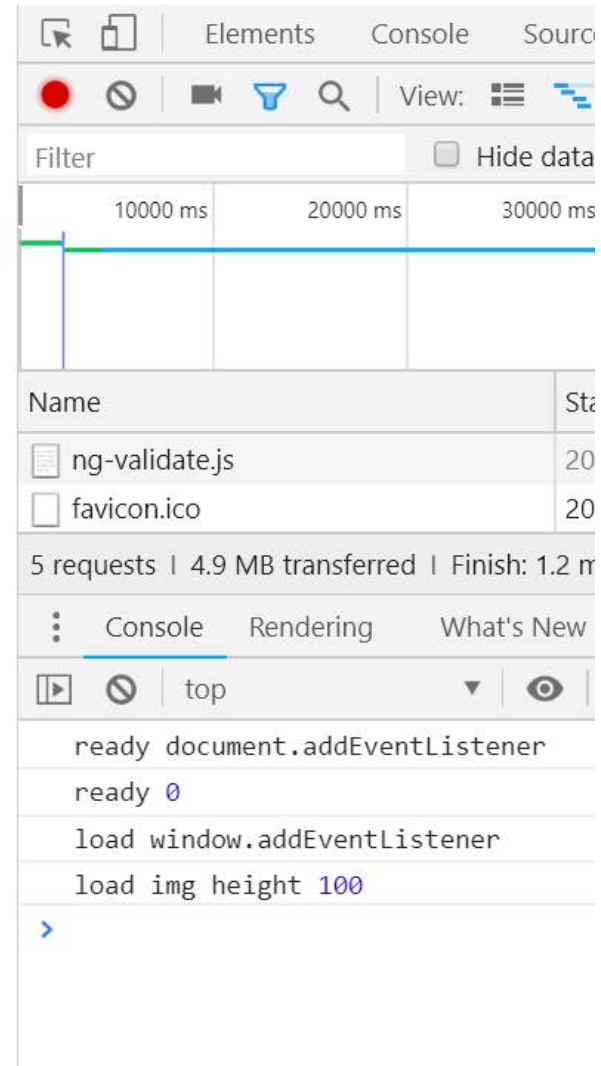
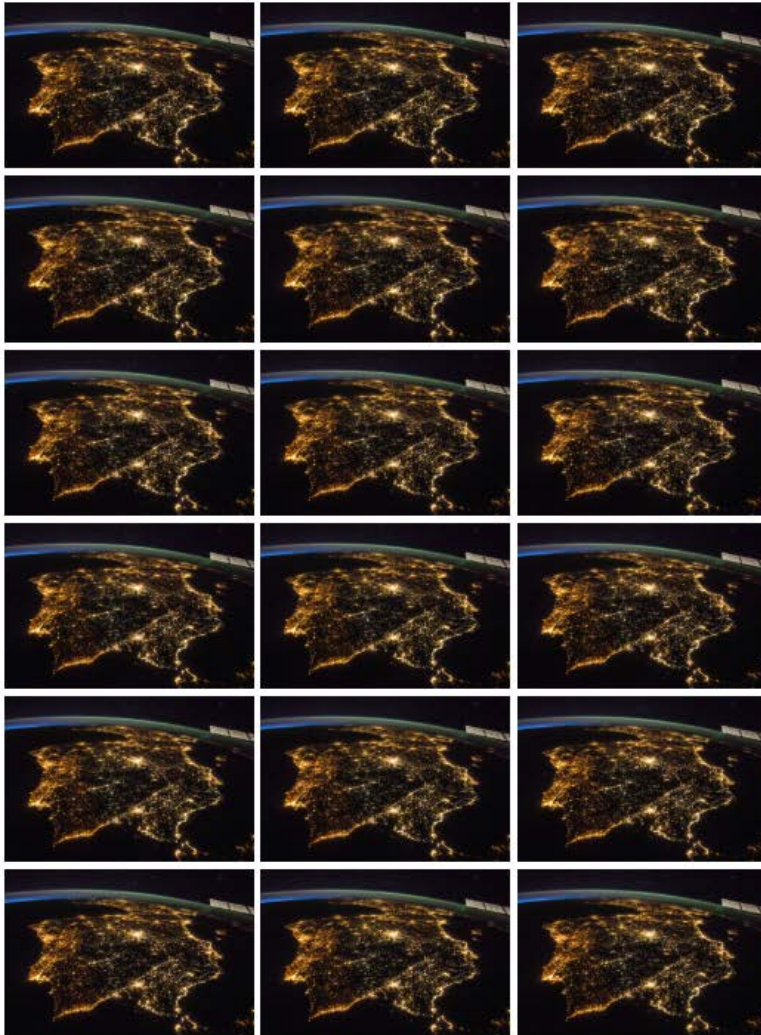
■ Soll auf «DOMContentLoaded» gewartet werden?

- Nötig für Libraries oder Skripts, die nicht wissen, wie sie genutzt werden.
- Nicht nötig für eigene Scripts. Diese am Ende vom Body einbinden.

■ Soll auf «Load» gewartet werden?

- Nötig, falls aufgrund von Elementgrößen Aktionen ausgelöst werden müssen z.B. platzieren von Elementen
- Selten nötig.

LOAD vs. DOMContentLoaded



defer / async

■ Script-Tags im Body sind nicht ideal

- Für ältere Browser die beste Variante.

■ Script Tags können mit defer oder async markiert werden

- Funktionieren nicht mit Inline-Skripts
- Falls beide verwendet werden, gewinnt async
- Hinweis: Bei alten Browser sehr unterschiedliches Verhalten

■ defer

- Die Skript Files sollen erst nach dem Parsen vom Dokument ausgeführt werden aber vor den «DOMContentLoaded»-Event.
- Reihenfolge der Files wird gewährleistet

■ async

- Die Skript-Files sollen asynchron geladen werden aber vor dem «load»-Event.
- Reihenfolge der Files wird nicht gewährleistet.

Defer / async

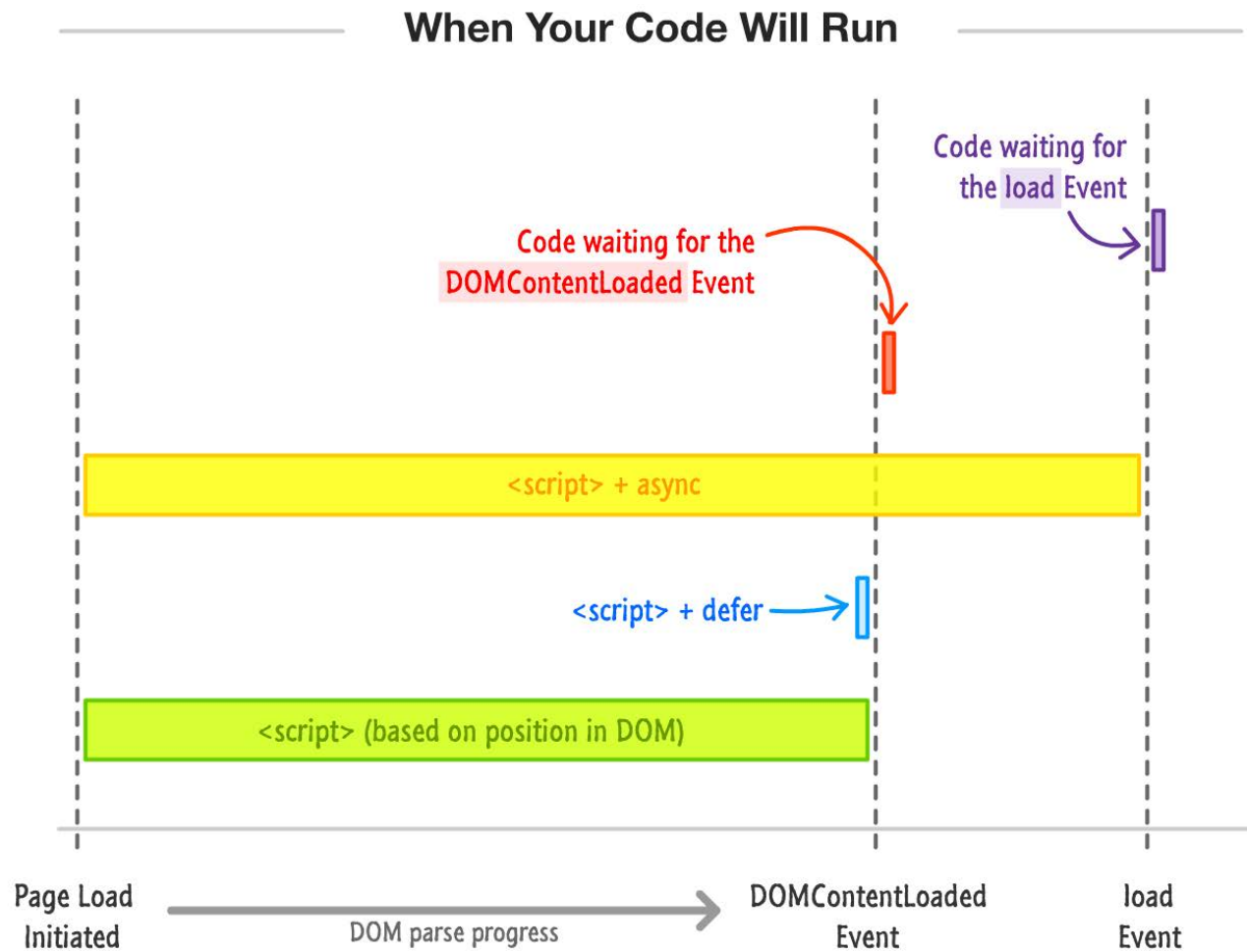
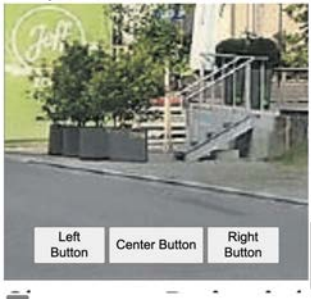


Bild-Quelle: https://www.kirupa.com/html5/running_your_code_at_the_right_time.htm

Kenntniss des Lifecycle relevant beim "Bug Hunting" / "Bug Fixing"

Beispiel Bug: Fehlender "Reflow" von Bild-Container in einem Bereich mit overflow: scroll

Layout ohne Update nach window.onload



Layout mit
`updateScrollableContainersSize`
registriert als `window.onload`-Handler



```
<style>
  .viewport {
    width: 50vw;
    height: 50vh;
    overflow: scroll;
  }
  .container-img-with-buttons {
    display: grid;
    grid-template-rows: 1fr auto;
    grid-template-columns: 1fr 1fr 1fr;
  }
  .container-img-with-buttons img {
    grid-row: 1 / 3;
    grid-column: 1 / 4;
  }
  .container-img-with-buttons button {
    grid-row: 2 / 3;
    margin-bottom: 30px;
  }
  .left-button {
    grid-column: 1 / 2;
    justify-self: left;
    margin-left: 30px;
  }
  .center-button {
    grid-column: 2 / 3;
    justify-self: center;
  }
  .right-button {
    grid-column: 3 / 4;
    justify-self: right;
    margin-right: 30px;
  }
</style>
```

```
<div class="viewport">
  <div class="container-img-with-buttons">
    
    <button class="left-button">Left Button</
    <button class="center-button">Center Butt
    <button class="right-button">Right Button
  </div>
</div>
```

```
<script>
  function updateScrollableContainersSize() {
    const containers =
      document.querySelectorAll('.container-img-with-b
    for (const container of containers) {
      const img = container.querySelector('img');
      container.style.width = img.width + 'px';
      container.style.height = img.height + 'px';
    }
  }

  window.addEventListener('load', updateScrollableConta
</script>
```

03-demoPicWithButtonsInViewportsGrid.html

DATA-* ATTRIBUTE

Data-*

- **HTML Elemente haben fest definierte Attribute.**
- **Benutzerdefinierte Attribute müssen mit dem Prefix «data-» beginnen.**
 - Hinweis: Der Browser toleriert vieles, der W3C Validator ist weniger tolerant.
- **Kann benutzt werden um Daten zwischen HTML und JavaScript auszutauschen.**
- **Die Werte werden als «Dictionary» im Property «dataset» abgelegt.**
 - Für alte Browser: `getAttribute()` / `setAttribute()` nötig
 - Der Name wird in «kebab-case» im HTML angegeben und im Dictionary als camelCase wiedergegeben
z.B. `data-factory-id` => `dataSet.factoryId`
- **Die Werte können via CSS ausgelesen werden.**
- Zum nachlesen: https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

JavaScript: Data-*

```
<article id="electriccars"
  data-columns="3"
  data-index-number="12314"
  data-parent="cars">

</article>
<script>
  console.log(document.getElementById("electriccars").dataset);
</script>
```

```
▼ DOMStringMap {columns: "3", indexNumber: "12314", parent: "cars"} ⓘ
  columns: "3"
  indexNumber: "12314"
  parent: "cars"
```

CSS: Data-*

- Mit `attr(...)` kann der Inhalt ausgelesen werden
- Attribute-Selektoren können die Werte selektieren

```
article::before {  
    content: attr(data-parent);  
}  
article[data-columns='3'] {  
    width: 400px;  
}  
article[data-columns='4'] {  
    width: 600px;  
}
```

Fazit: Data-*

- Verwenden um Meta-Informationen bei HTML-Elementen zu hinterlegen.
Z.B.

- Welche ID betrifft es
- Welche Aktion wird ausgelöst

- Nützlich für Event-Bubbling

```
const container = document.getElementById("container");

container.addEventListener("click", (event) => {
  const id = event.target.closest(".js-item-container").dataset.id;
  if (args.target.classList.contains("js-delete")) {
    console.log("remove", id);
    event.stopPropagation();
  }
  else if (args.target.classList.contains("js-add")) {
    console.log("add", id);
    event.stopPropagation();
  }
});
```


ÜBUNG

Event Bubbling

- Es soll möglich sein die Songs zu bewerten.
 - Die Songs sollen nach ihrem Rating sortiert gerendert werden.
 - Nutzen Sie Event Bubbling und data-*
-
- Vorlage: 99_Uebungen/Vorlagen/04_EventBubbling.html
 - songmodel.js: Beinhaltet die Hilfsfunktionen und die Daten.
 - Lösung: 99_Uebungen/solution

Songs

- 3 Thank you for the music
ABBA
- 2 California Girls
Beach Boys
- 1 How Deep Is Your Love

TEMPLATING

Motivation

- Sehr grosse Code-Menge nötig im JavaScript um DOM-Elemente zu erstellen
- Bei grösseren HTML-Fragmenten leidet die Übersichtlichkeit

```
const df = document.createDocumentFragment();
for (const song of songs) {
  const liElement = document.createElement("li");
  const h3Element = document.createElement("h3");
  liElement.appendChild(h3Element);
  h3Element.textContent = song.title;
  const pElement = document.createElement("p");
  liElement.appendChild(pElement);
  pElement.textContent = song.artist;
  df.appendChild(liElement);
}
document.getElementById("songs").appendChild(df);
```



```
<li>
  <h3>Thank you for the music</h3>
  <p>ABBA</p>
</li>
```

Lösung: Template-Engine

- **Template-Engines ermöglichen Darstellungs-Logik von Programme-Code zu trennen**
- **SPA-Frameworks kommen mit einer Template Engine**
- **Express.js verwendet eine Template Engine für Server-Side-Rendering**
- **Diese ermöglichen es Templates zu definieren und diese bei Gebrauch mit Daten zu rendern.**
- **Handlebars (<https://handlebarsjs.com/>) ist eine simple standalone Template-Engine**
- **Alternativen**
 - Pug
 - Jade
 - Mustache
 - Template Strings

Handlebars

```
<!--Das Template kann direkt im HTML definiert werden.-->
<script id="song-template" type="text/x-handlebars-template">
  {{#each this}}
    <li>
      <h3>{{title}}</h3>
      <p>{{artist}}</p>
    </li>
  {{/each}}
</script>
```

```
// auslesen des Templates
const songsFragmentTemplateSource = document.getElementById("song-template").innerHTML;

// Template ist nur ein String, durch das compile() wird daraus eine Template-Funktion
const createSongsFragmentHtmlString = Handlebars.compile(songsFragmentTemplateSource);

function renderSongs() {
  //die Template-Funktion kann aufgerufen werden. Als Parameter werden die Daten übergeben.
  document.getElementById("songs").innerHTML = createSongsFragmentHtmlString(songs);
}
```

Handlebars

- Bei jeder Template Engine können dem Template Daten übergeben werden.
 - Bei Handlebars wird der Parameter zum (Root)-Context vom Template
Der Root-Context wird im Template mit `{{this}}` angesprochen.
Properties des Context mit `{{this.property}}` oder `{{property}}`

```
<script id="entry-template" type="text/x-handlebars-template">  
  <figure>
```

...

```
    <figcaption>  
      <p>{{this.description}}</p>  
    </figcaption>  
  </figure>  
</script>
```

Loops

■ Bei Loops wird das aktuelle Objekt als Sub-Context definiert.

- Zugriff auf den aktuellen Index: `{{@index}}`
- Zugriff auf Root: `{{@root}}`

```
<ul>
  {{#each items}}
  <li>
    <h3>{{title}}</h3>
    <p id="item-{{@index}}">{{@root.name}}: {{this.artist}}</p>
  </li>
  {{/each}}
</ul>
```


Helpers

■ Handlebars definiert nur wenige eigene Statements:

#if	#lookup
#unless	#log
#each	#blockHelperMissing
#with	#helperMissing

■ Es können eigene «Helper» definiert werden: https://handlebarsjs.com/block_helpers.html

```
Handlebars.registerHelper('?', function(exp, value1, value2, options) {  
    if(exp) {  
        return value1;  
    }  
    return value2;  
});
```

```
<script id="template" type="text/x-handlebars-template">  
    {{? hasError 'FEHLER' 'OK' }}  
</script>
```

Helpers: Block

```
// Usage: {{#list elements}}<span>{{elementProp}}</span>{{/list}}
Handlebars.registerHelper('list', function(context, options) {
  let out = '<ul>';

  Array.from(context).forEach((ctx) => {
    out += '<li>' + options.fn(ctx) + '</li>';
  });
  return out + '</ul>';
});
```

Bei Block-Helpers enthält das options Argument den "Body" des Blocks und options.fn die daraus erstellte Template-Funktion. Mit options.fn(ctx) wird die Template-Funktion auf den Context ctx angewendet.

```
<script id="template" type="text/x-handlebars-template">
  {{#list elementsToList}}
    <span>{{title}}</span>
  {{/list}}
</script>
```

ÜBUNG

Handlebars

- Nutzen Sie Handlebars und Eventbubbling um die Song Liste zu rendern

Songs

- 3 **Thank you for the music**

ABBA

- 2 **California Girls**

Beach Boys

- 1 **How Deep Is Your Love**

- Vorlage: 99_Uebungen/Vorlagen

- 05_Handlebars.html oder die eigene Lösung von Aufgabe 04_EventBubbling verwenden
- songmodel.js: Beinhaltet die Hilfsfunktionen und die Daten.

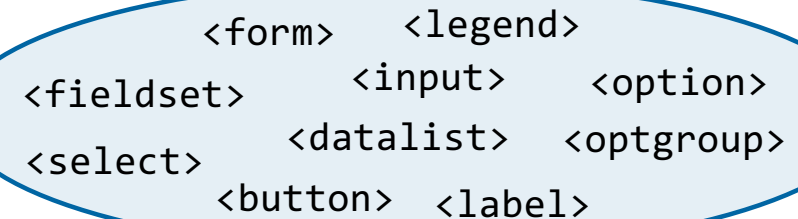
- Lösung: 99_Uebungen/solution

FORM VALIDATION

Formular <form>

■ Besteht aus

- Formular <form>
 - Group of Form Fields <fieldset>
 - Caption for Content <legend>
- Textarea <textarea>
- Selection of Options <select>
 - Option <option>
 - Option Group <optgroup>
- Label <label>
- Button <button>
- Input Element <input>
 - Lists pre-defined values <datalist>
- <progress> / <ouput> / <meter> / <keygen> / ...



<form> <legend>
<fieldset> <input> <option>
<select> <datalist> <optgroup>
<button> <label>

bekannt

Formular <form> + sub-Elemente

■ form-Element

- Notwendig um User Input ohne JS zum Server zu schicken (mittels Form-Submission)
- Ermöglicht «Submit on Enter» Verhalten

■ Wichtige form Attribute

- action (URL zu der die Daten geschickt werden)
- method
 - get: Daten werden in der URL/Query-String gesendet..
 - post: Daten werden im Request-Body gesendet

■ Form Sub-Elemente im Beispiel rechts:

(auch ohne umgebendes Form nutzbar mit JavaScript)

- label-Element (Anschrift)
- input-Element (Input von Text, Zahl, Auswahl)
- button-Element

■ Generelle Attribute von Form Sub-Elementen

- name bestimmt Feldnamen in Requ. an Server
- required bestimmt ob Feld gefüllt werden muss (boolean)

11-forms/01-firstForm.html

bekannt

```
<h1>First Form</h1>
<form action="01b-firstFormSubmit.html">
  <label for="message"><span>Message: </span>
    <input id="message" name="message"
      type="text"
      placeholder="Your Message Here"
      required>
  </label>
  <button>Send Message</button>
</form>
```

id nötig für label

HTML Formulare: Traditionelle Browser <-> Server Interaktion

1. Browser erhält Formular und stellt es dar
2. Nutzer füllt Informationen aus (Visuelles Feedback zu valid/invalid Elements)
3. Nutzer drückt Submit
 - a) Submission von invalidem Formular wird verhindert. Fehlermeldung für das erste invalide Element angezeigt
 - b) Formular-Information wird an Server gesendet
 - post: im Body,
 - get: in Query-String
4. Server nimmt Informationen entgegen und sendet Antwortseite
5. Browser stellt Antwortseite dar

1

First Form

Message:

2

First Form

Message:

3a

First Form

Message: 

Please fill in this field.

3b

```
method: GET;  
url: /submit?message=hello+form;  
query: message=hello+form;  
body:
```

4+5

Form Submit Server Response

Thank you for sending your message ...

Deklarative Validierung von Formularen

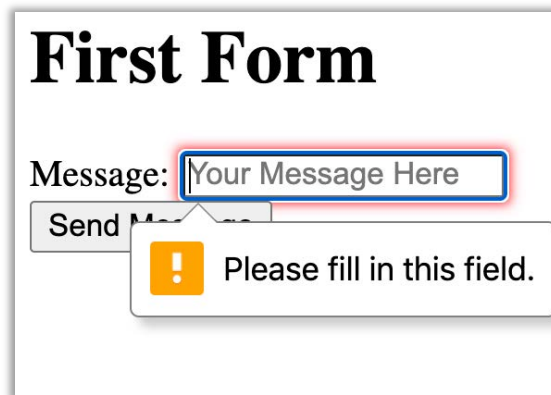
Input Felder können definieren wie korrekt Informationen für sie aussehen sollen

Invalide Felder können durch CSS (`input:valid { ... }`) markiert werden

Submission eines Formulars mit invaliden Feldern wird defaultmässig vom Browser verhindert

Beispiele

- `<input ... required>` -> Eingabe ist nötig
- `<input type=email>` -> E-Mail erwartete (String mit @)
- `<input pattern="[A-Za-z0-9]+">` nur "einfache" Buchstaben und Zahlen, kein Blank erlaubt
- `<input type=number>` -> nur Zahlen (häufig andere Eingabe nicht möglich)
- `<input type=date>` -> nur Datum (häufig andere Eingabe nicht möglich)



The screenshot shows a web form titled "First Form". It contains a label "Message:" followed by a text input field with the placeholder text "Your Message Here". The input field is highlighted with a red border, indicating it is required. Below the input field is a "Send Message" button. A validation error message is displayed below the button, consisting of an orange square icon with a white exclamation mark and the text "Please fill in this field."

Form Validation mit JavaScript

- **Die deklarative Validation unterschützt nur limitiert Anpassungen:**
 - Die Fehlermeldung ist vom Browser vorgegeben.
 - Das Aussehen der Fehlermeldung (Styling) ist vorgegeben.
- **Abhängigkeiten können nicht überprüft werden.**
 - Beispiel: Bei verheirateten Paaren ist der Name vom Partner/-in auch einzugeben.

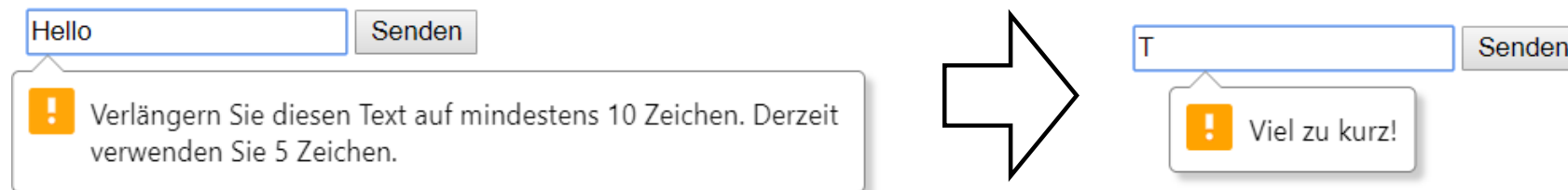
Die deklarative Validation (Regex, Pattern, Type) deckt nicht alle Szenarios ab:

-> Scripting erforderlich

Erstes Beispiel: Fehlermeldungstexte anpassen

■ Der Standardtext ist ohne JavaScript nicht anpassbar.

- Lokalisierung
- Bessere Fehlermeldung



■ Fehlermeldungstexte mit JS anpassen

1. Auf den Status-Wechsel reagieren (EventListener registrieren)
2. Den Fehler-Status auslesen (tooShort)
3. Den Fehlermeldungstext anpassen

Erstes Beispiel: Fehlermeldungstexte anpassen

```
<h1>Custom Validation For Demo</h1>
<form>
  <label for="value">Value:</label>
  <input id="value" minlength="10" required>
  <input type="submit">
</form>
```

11-forms/
02_custom-validation.html

(1) Jedes FormElement besitzt das validity-Property.

```
<script>
  const value = document.getElementById("value");

  value.addEventListener("input", function (event) {
    if (value.validity.tooShort) {
      value.setCustomValidity("Viel zu kurz!");
    } else {
      value.setCustomValidity("");
    }
  });
</script>
```

(2) HTML5-Validation wird als Property repräsentiert

```
▼ ValidityState {valueMissing: true,
  badInput: false,
  customError: false,
  patternMismatch: false,
  rangeOverflow: false,
  rangeUnderflow: false,
  stepMismatch: false,
  tooLong: false,
  tooShort: false,
  typeMismatch: false,
  valid: false,
  valueMissing: true}
```

(4) Fehlermeldung / Fehlerzustand rücksetzen

(3) Es ist möglich eigene Fehlermeldungen zu definieren.

Validation API

Constraint validation API properties

Property	Description
<code>validationMessage</code>	A localized message describing the validation constraints that the control does not satisfy (if any), or the empty string if the control is not a candidate for constraint validation (<code>willValidate</code> is <code>false</code>), or the element's value satisfies its constraints.
<code>validity</code>	A <code>ValidityState</code> object describing the validity state of the element. See that article for details of possible validity states.
<code>willValidate</code>	Returns <code>true</code> if the element will be validated when the form is submitted; <code>false</code> otherwise.

Validation API

Constraint validation API methods

Method	Description
<code>checkValidity()</code>	Returns <code>true</code> if the element's value has no validity problems; <code>false</code> otherwise. If the element is invalid, this method also causes an <code>invalid</code> event at the element.
<code>HTMLFormElement.reportValidity()</code>	Returns <code>true</code> if the element's child controls satisfy their validation constraints. When <code>false</code> is returned, cancelable <code>invalid</code> events are fired for each invalid child and validation problems are reported to the user.
<code>setCustomValidity(<i>message</i>)</code>	<p>Adds a custom error message to the element; if you set a custom error message, the element is considered to be invalid, and the specified error is displayed. This lets you use JavaScript code to establish a validation failure other than those offered by the standard constraint validation API. The message is shown to the user when reporting the problem.</p> <p>If the argument is the empty string, the custom error is cleared.</p>

Beispiel: Fehlermeldung anders darstellen

Ziel: Die Standardfehlermeldung durch eine eigene Fehlermeldung ersetzen.

1. Standardfehlermeldung verbergen mit novalidate

- Hinweis: Submit überprüft nicht mehr ob die Form Valid ist.

```
<form novalidate>
  <input id="value" minlength="10" required>
  <input type="submit">
</form>
```

2. Eigene Fehlermeldung darstellen.

- Beispiel:
 - <https://codepen.io/anon/pen/WWGLxR?&editable=true>
 - <https://validatejs.org/examples.html>

HTML 5 Constraint Validation API: Beispiel: Custom Validation + Custom Display

Custom Validation + Error Display

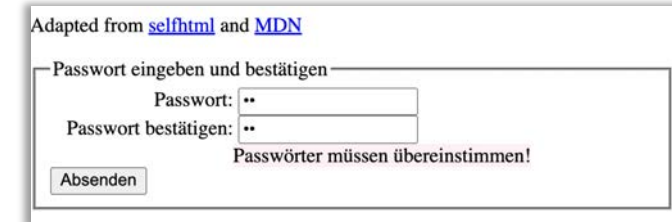
- (1) Mit dem form-Attribut **novalidate** wird die automatische Validierung abgestellt.
- (2) Validierung kann mit **checkValidity()** explizit angestoßen werden
- (3) Fehlermeldungen können durch reguläre DOM Manipulationen dargestellt werden (z.B. ****)
- (4) Im Fall von Fehlern muss das Absenden des Formulars explizit durch Aufruf von **event.preventDefault();** verhindert werden.

```

1 <form novalidate>
  <fieldset>
    ...
    <label for="password2"><span>Passwort bestätigen:</span>
      <input type="password" required id="password2">
      <span id="pw-error" aria-live="polite"></span> <!--no content -> not visible-
    </label>
    <button>Absenden</button>
  </fieldset>
</form>
<script>
  ...
  const checkPasswordValidity = function (event) {
2    formElement.checkValidity();
    if ((!password1Element.validity.valid) || (!password2Element.validity.valid)) {
3      errorElement.innerText = 'Beide Passwörter müssen eingegeben werden!';
      event.preventDefault();
    } else {
      if (password1Element.value !== password2Element.value) {
4        errorElement.innerText = 'Passwörter müssen übereinstimmen!';
        event.preventDefault();
      } else {
        errorElement.innerText = '';
      }
    }
  };
  formElement.addEventListener('submit', checkPasswordValidity);
  password2Element.addEventListener('change', checkPasswordValidity);

```

03b-customValidationNovalidate2.html



Beispiel (adaptiert von https://wiki.selfhtml.org/wiki/JavaScript/Tutorials/Formulareingaben_mit_JavaScript_validieren)

HTML 5 Constraint Validation API: Beispiel: Custom Validation + Custom Display + Dirty

Custom Validation

+ Error Display

+ Dirty-Handling

- Ziel: Validity-Status nur der schon besuchten Feldern anzeigen

- (1) Methode: Input-Felder beim Verlassen (onblur) mit der Klasse **.dirty** kennzeichnen.

■ Felder mit CSS stylen:

- (2) "Dirty" Felder die aktuell nicht mehr bearbeitet werden (nicht mehr den Fokus haben), erhalten mittels CSS einen roten Hintergrund (plus weitere Kennzeichnung) wenn sie nicht korrekt ausgefüllt wurden.
- (3) "Dirty" Felder die aktuell nicht mehr bearbeitet werden, erhalten mittels CSS einen grünen Hintergrund (plus weitere Kennzeichnung) wenn sie korrekt ausgefüllt wurden.

```
<style>
```

```
2 input.dirty:not(:focus):invalid {
  background-color: #FFD9D9; /* red /* ...
}
3 input.dirty:not(:focus):valid {
  background-color: #D9FFD9; /* green /* ...
}
```

```
</style>
```

```
</head>
```

```
<form novalidate>
```

```
</form>
```

```
<script>
```

```
const checkPasswordValidity = function (event) {
```

```
...
};
```

```
formElement.addEventListener('submit', checkPasswordValidity);
```

```
password2Element.addEventListener('change', checkPasswordValidity);
```

```
1 const setDirty = function (event) {
```

```
  event.target.classList.add("dirty");
```

```
};
```

```
password1Element.addEventListener('blur', setDirty);
```

```
password2Element.addEventListener('blur', setDirty);
```

```
</script>
```

03c-customValidationNovalidatePlusDirty.html

Adapted from [selfhtml](#) and [MDN](#) and [developers.google](#)

Passwort eingeben und bestätigen

Passwort:

Passwort bestätigen:

Beide Passwörter müssen eingegeben werden!

Beispiel (adaptiert von <https://developers.google.com/web/fundamentals/design-and-ux/input/forms/>)

Achtung: Statische Referenzen sind gut, ABER

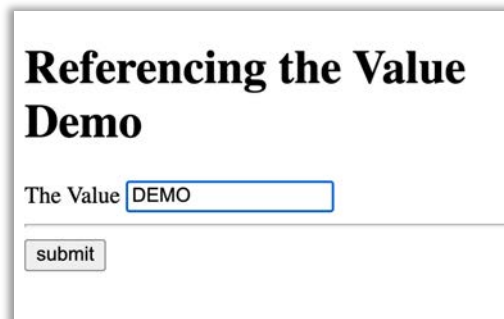
```
<body>
<h1>Referencing the Value Demo</h1>

<label for="inpt">The Value</label>
<input id="inpt" name="thevalue" value="the default value">
<hr>
<button id="btn">submit</button>

<script>
  const theValue = document.querySelector("#inpt").value;
  const theButton = document.querySelector("#btn")
  theButton.addEventListener('click',
    () => console.log('the value is: ', theValue));
</script>
```

Was ist der Output in der Konsole wenn "submit" gedrückt wird?

- A: DEMO
- B: undefined
- C: the default value
- D: Kein Output / keine Reaktion
- E: Error



Referencing the Value Demo

The Value

Validation API Links

- https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation
- <https://developer.mozilla.org/en-US/docs/Web/API/ValidityState>

- **Anleitungen:**
 - https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation
 - <https://css-tricks.com/form-validation-part-2-constraint-validation-api-javascript/>

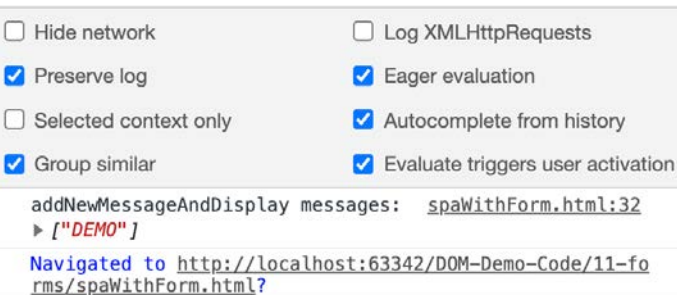
Formulare in Single-Page Apps – Vorteil: Submit on Return. Problem? Wie lösen?

```
<h1>SPA with Form</h1>
<ul id="messages-list"></ul>
<form>
  <label for="new-message-input">
    New Message: </label>
  <input id="new-message-input">
  <button>Add Message</button>
</form>
```

SPA with Form

- No messages yet. Enter below your first message

New Message:



```
<script>
  const newMessageInput = document.getElementById('new-message-input');
  const messagesList = document.getElementById('messages-list');
  const submitButton = document.querySelector('button');

  const messages = [];

  function getMessagesHTMLString() {
    if (messages.length > 0) {
      return messages.map(message => `<li>${message}</li>`).join('');
    } else {
      return '<li>No messages yet. Enter below your first message</li>';
    }
  }

  function addNewMessageAndDisplay() {
    messages.push(newMessageInput.value);
    console.log('addNewMessageAndDisplay messages: ', messages);
    messagesList.innerHTML = getMessagesHTMLString();
  }

  submitButton.addEventListener('click', addNewMessageAndDisplay);

  messagesList.innerHTML = getMessagesHTMLString();
</script>
```

11-forms/04-spaWithForm-Problem.html

Formulare in Single-Page Apps –

Statt `<button>.onclick` `<form>.onsubmit` nutzen + `event.preventDefault()`

141
DOM

```
<h1>SPA with Form - Solved</h1>
<ul id="messages-list"></ul>
<form>
  <label for="new-message-input">
    New Message: </label>
  <input id="new-message-input">
  <button>Add Message</button>
</form>
```

SPA with Form - Solved

- DEMO
- DEMO2
- DEMO3

New Message:

```
const newMessageInput = document.getElementById('new-message-input');
const messagesList = document.getElementById('messages-list');
const newMessageForm = document.querySelector('form');
```

```
const messages = [];
```

11-forms/04b-spaWithForm-Better.html

```
function getMessagesHTMLString() {
  if (messages.length > 0) {
    return messages.map(message => `<li>${message}</li>`).join('');
  } else {
    return '<li>No messages yet. Enter below your first message</li>';
  }
}

function addNewMessageAndDisplay() {
  messages.push(newMessageInput.value);
  console.log('addNewMessageAndDisplay messages: ', messages);
  messagesList.innerHTML = getMessagesHTMLString();
}

newMessageForm.addEventListener('submit', event => {
  event.preventDefault();
  addNewMessageAndDisplay();
});

messagesList.innerHTML = getMessagesHTMLString();
```

Übung Custom Form Validation

■ Vorlage:

Vorlagen > UebungFormValidation > FormularAufgabe.html

- Fehlercheck in den .js Files setzen
- Leerfelder (@@____ @@) ausfüllen

■ Lösung: 99-Uebungen > solution > UebungFormValidation > x-FormularAufgabeLoesung.html

Form Validation Aufgabe

Enter your Credit Card Information

Name on Card

Number

Expiration MM-YY

Form Validation Aufgabe

Enter your Credit Card Information

Name on Card

Number

Expiration MM-YY

! Please fill in this field.

Form Validation Aufgabe

Enter your Credit Card Information

Name on Card

Number

Expiration MM-YY

! Error in Card Number

STORAGE

Storage

■ LocalStorage / SessionStorage

- Speichern von String-Werten
- Objekte mit `JSON.stringify({ })` möglich.
- Bzw.: `JSON.parse("{ }")`

■ Usage:

- `localStorage.setItem("key", "value")`
- `localStorage.getItem("key")`

■ <https://developer.mozilla.org/en-US/docs/Web/API/Storage>

■ <https://html.spec.whatwg.org/multipage/webstorage.html>

- A mostly arbitrary limit of five megabytes per origin is suggested

■ <https://web.dev/storage-for-the-web/>

- For the network resources necessary to load your app and file-based content, use the Cache Storage API (part of service workers).
- For other data, use the async IndexedDB (with a promises wrapper).

LocalStorage vs. SessionStorage

- **Gleiche API**

- **LocalStorage**

- Speichert die Daten bis diese gelöscht werden
- Geshared über Tabs und Browser-Fenster

- **SessionStorage**

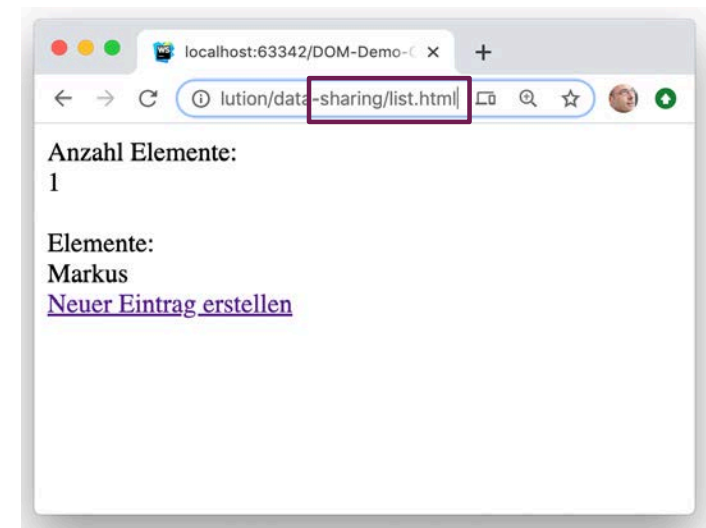
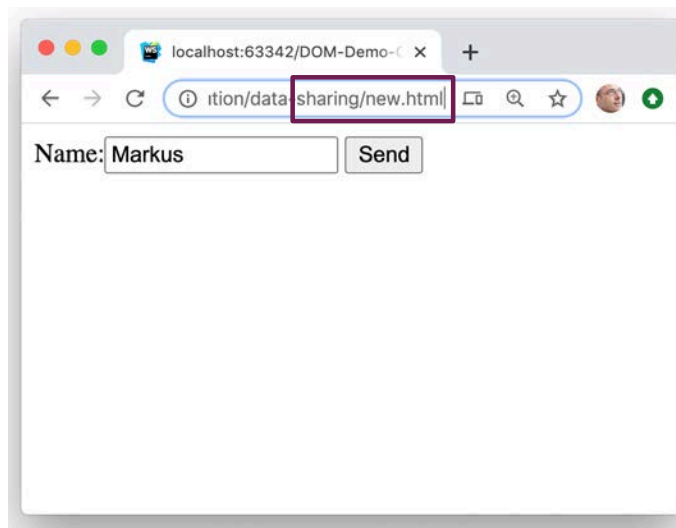
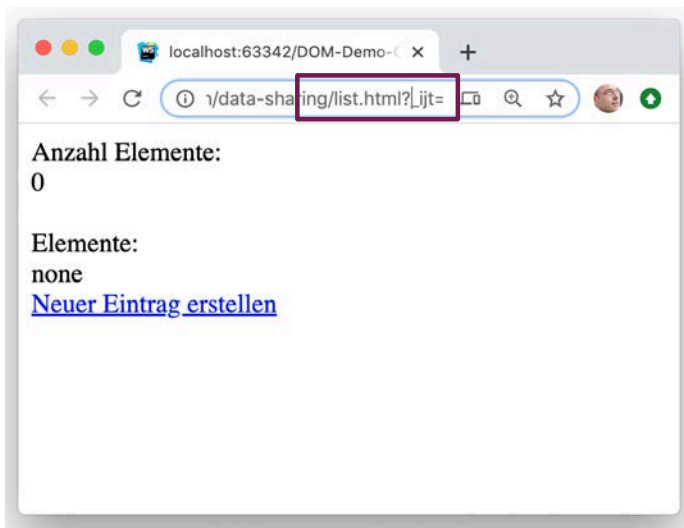
- Privat pro Tab
- Flüchtig: Bei Tab / Window geschlossen werden die Daten verworfen
- Bei einem Refresh überleben die Daten

The screenshot shows the Chrome DevTools Resources panel. The left sidebar lists various resource categories: Frames, Web SQL, IndexedDB, Local Storage, Session Storage, Cookies, Application Cache, Cache Storage, and Service Workers. Under Session Storage, the entry 'http://localhost:63342' is selected and highlighted in blue. The main panel displays a table with two columns: 'Key' and 'Value'. The first row shows the key 'users' and the value '["Michael"]'.

Key	Value
users	["Michael"]

Aufgabe: Daten zwischen HTML Pages sharen

- Ausgangslage: 99_Uebungen/Vorlagen/data-sharing
- Aufgabe: Erstellen einer Master-Detail Ansicht.
- Beispiellösung: 99_Uebungen/solution/data-sharing



REFERENZEN

Referenzen

- https://wiki.selfhtml.org/wiki/JavaScript/Tutorials/Grundlagen_des_DOM
- https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation
- <https://handlebarsjs.com/>

Weiterführende Links

- Abschnitt DOM Tools in der Liste von Frontend Masters
<https://frontendmasters.com/books/front-end-handbook/2019/#6.10>
- The Event Loop (Jake Archibald) --
<https://youtu.be/cCOL7MC4PI0?t=109> (30»)
- Life of a Pixel (HTML Parsing and Rendering in Depth)
<https://bit.ly/lifeofapixel>
- Inside look at modern web browser (Part 1 – 4)
<https://developers.google.com/web/updates/2018/09/inside-browser-part1>
- JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language 7th Edition, Kindle Edition (July 2020)
https://www.amazon.com/JavaScript-Definitive-Most-Used-Programming-Language-ebook-dp-B088P9Q6BB/dp/B088P9Q6BB/ref=mt_kindle?_encoding=UTF8&me=&qid=

JavaScript: The Definitive Guide: Master the W
7th Edition, Kindle Edition
by David Flanagan (Author) Format: Kindle Edition

